AFRL-IF-WP-TR-2001-1543

DESIGN TOOLS AND ARCHITECTURES FOR DEDICATED DIGITAL SIGNAL PROCESSING (DSP) PROCESSORS

Keshab K. Parhi

University of Minnesota
200 Union Street SE
Minneapolis, MN 55455

July 1996

FINAL REPORT FOR PERIOD 24 AUGUST 1993 – 24 AUGUST 1996

Approved for public release; distribution unlimited.

20020103 134

INFORMATION DIRECTORATE
AIR FORCE RESEARCH LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7334

NOTICE

USING GOVERNMENT DRAWINGS, SPECIFICATIONS, OR OTHER DATA INCLUDED IN THIS DOCUMENT FOR ANY PURPOSE OTHER THAN GOVERNMENT PROCUREMENT DOES NOT IN ANY WAY OBLIGATE THE US GOVERNMENT. THE FACT THAT THE GOVERNMENT FORMULATED OR SUPPLIED THE DRAWINGS, SPECIFICATIONS, OR OTHER DATA DOES NOT LICENSE THE HOLDER OR ANY OTHER PERSON OR CORPORATION; OR CONVEY ANY RIGHTS OR PERMISSION TO MANUFACTURE, USE, OR SELL ANY PATENTED INVENTION THAT MAY RELATE TO THEM.

THIS REPORT IS RELEASABLE TO THE NATIONAL TECHNICAL INFORMATION SERVICE (NTIS). AT NTIS, IT WILL BE AVAILABLE TO THE GENERAL PUBLIC, INCLUDING FOREIGN NATIONS.

THIS TECHNICAL REPORT HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION.

LUIS M. CONCHA
Team Leader
Collaborative Simulation Technology Branch
Information Systems Division
Information Directorate

G. Todd Berry
Acting Chief
Collaborative Simulation Technology Branch
Information Systems Division
Information Directorate

WALTER B. HARTMAN
Acting Chief
Wright Site
Information Directorate

Do not return copies of this report unless contractual obligations or notice on a specific document requires its return.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** JULY 1996	**3. REPORT TYPE AND DATES COVERED** FINAL,08/24/93 -- 08/24/96

4. TITLE AND SUBTITLE
Design Tools and Architectures for Dedicated Digital Signal Processing (DSP) Processors

5. FUNDING NUMBERS
C: F33615-93-C-1309
PE 63739E
PR A268
TA 02
WU 03

6. AUTHOR(S)
Keshab K. Parhi

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

University of Minnesota
200 Union Street SE
Minneapolis, MN 55455

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Information Directorate
Air Force Research Laboratory
Air Force Materiel Command
Wright-Patterson AFB, OH 45433-7334
POC: Luis Concha, AFRL/IFSD, 51901 x3578

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

AFRL-IF-WP-TR-2001-1543

11. SUPPLEMENTARY NOTES
None

12a. DISTRIBUTION AVAILABILITY STATEMENT
Approved for public release, distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The work reported in this document is concerned with the development of CAD tools, design methodologies, and architectures for the following topics of VLSI digital signal processing: high-level transformations and synthesis, discrete wavelet transform, high-speed digital subscriber loops, and finite field arithmetic for use in Reed-Solomon coders. Through this research we developed fast and efficient algorithms, ILP models, and tools that would reduce the time to explore the design space and locate an area optimal design of ASICs for DSP applications within a heterogeneous environment.

14. SUBJECT TERMS
RASSP, discrete wavelet transform, high speed digital subscriber loops

15. NUMBER OF PAGES
116

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

Standard Form 298 (Rev. 2-89) (EG)
Prescribed by ANSI Std. 239.18
Designed using Perform Pro, WHS/DIOR, Oct 94

Contents

1 Introduction 2

CAD Tools 3

2 High-Level Synthesis 3
- 2.1 MARS Design Tool 5
  - 2.1.1 MARS overview 6
  - 2.1.2 Loop-Based Synthesis 6
  - 2.1.3 Module Selection 8
  - 2.1.4 Scheduling 10
  - 2.1.5 Experimental Results 12
- 2.2 Integer Linear Programming High-Level Synthesis 16
  - 2.2.1 Time-Constrained Scheduling by ILP 16
  - 2.2.2 Counting the Number of Registers During Scheduling 18
  - 2.2.2.1 The Models of Processors and Registers 18
  - 2.2.2.2 The Technique to Count the Number of Registers 19
  - 2.2.2.3 The Number of Registers in Overlapped Schedule 20
  - 2.2.2.4 Registers for Digit-Serial Data Architecture 23
  - 2.2.3 Register Minimization in Architectures with Multiple Data Formats . 26
  - 2.2.3.1 ILP Model for Processor Type Selection 26
  - 2.2.3.2 Counting the Number of Registers During the Time Assignment . . 29
  - 2.2.4 Experimental Result 32

3 Other High-Level Tools 38
- 3.1 Determination of Minimum Iteration Period 38
  - 3.1.1 A New Algorithm to Determine the Iteration Bound 39
  - 3.1.2 Experimental Results 42
- 3.2 Exhaustive Scheduling and Retiming 43
- 3.3 Two-Dimensional Retiming 45

Architectures 47

4 Discrete Wavelet Transforms 47
- 4.1 Multirate Folding 48
- 4.2 Register Minimization 49
- 4.3 Lattice-Based DWT Architectures 51
- 4.4 Architectures for Tree-Structured Filter Banks 53

5 High-Speed Digital Communications: HDSL/ADSL/VDSL 54
- 5.1 Background 56
  - 5.1.1 Motivation 56
  - 5.1.2 Discrete Multitone 57
  - 5.1.3 Carrierless AM/PM Transceiver 59

5.2 Three-Dimensional CAP 61
5.3 ODMA 63
5.4 Simulation Results 64
5.5 LMS Relaxation 65
5.6 Concluding Remarks 68

6 Finite Field Arithmetic and Reed-Solomon Coders **69**
6.1 Efficient Power Based Galois Field Arithmetic Architectures 70
6.1.1 Conversion to Power 70
6.1.2 Result Computation 71
6.1.3 Conversion to Conventional Basis 72
6.1.4 Comparison with Other Architectures 73
6.2 Low Latency Standard Basis $GF(2^m)$ Multiplier and Squarer Architectures . 74
6.2.1 Parallel-in-Parallel-out Multiplier 74
6.2.1.1 Multiplier Architecture 74
6.2.1.2 VLSI Chip Implementation 74
6.2.1.3 Comparison with Other Multipliers 75
6.2.2 Parallel-in-Parallel-out Squarer 76
6.2.2.1 Squarer Architecture 76
6.2.2.2 Comparison with Other Designs 78
6.2.3 Parallel-in-Parallel-out Exponentiator 79
6.2.3.1 Exponentiation Algorithm 79
6.2.3.2 Exponentiator Architecture 79
6.2.3.3 Architecture Comparison 81
6.3 Efficient Standard Basis Reed-Solomon Encoder 81
6.3.1 Reed-Solomon Encoding Algorithm 82
6.3.2 Reed-Solomon Encoder 82
6.3.3 Comparison with Berlekamp's Dual Basis RS Encoder 84
6.4 Efficient Finite Field Serial/Parallel Multiplication 84
6.4.1 Bit-Serial Finite Field Multiplier 84
6.4.1.1 Multipier Architecture 84
6.4.1.2 Comparison with Other Designs 86
6.4.2 Generalized Serial/Parallel Finite Field Multiplication 87
6.4.2.1 Digit-Serial Multiplication Algorithms 87
6.4.2.2 Multiplier over $GF(2^8)$ 90

7 Order-Configurable, Power Efficient FIR Filters **92**
7.1 Background 93
7.2 Configurable Processor Array 95
7.3 Phase Locked Loop 96
7.4 Simulation 97

8 List of Publications Supported by RASSP **98**

References **102**

Design Tools and Architectures for Dedicated DSP Processors

Keshab K. Parhi, Principal Investigator
Professor, Department of Electrical Engineering
University of Minnesota, Minneapolis, MN 55455
Tel: (612) 624-4116
Fax: (612) 625-4583
E-mail: parhi@ee.umn.edu
Web: http://www.ee.umn.edu/users/parhi

July 15, 1996

Abstract

In the past three years, we have addressed and developed CAD tools, design methodologies, and architectures for the following topics of VLSI digital signal processing: high-level transformations and synthesis, discrete wavelet transform, high-speed digital subscriber loops, and finite field arithmetic for use in Reed-Solomon coders. This report summarizes our results in these areas. Through this research we developed fast and efficient algorithms, ILP models, and tools that would reduce the time to explore the design space and locate an area optimal design of ASICs for DSP applications within a heterogeneous environment. In this project, the phrase "heterogeneous architectures" defines any architecture that contains different types of functional units (including algorithms and implementation styles) to process the same type operations. By utilizing a heterogeneous library, one removes the word size and implementation style restrictions and allows the system to explore a much wider design space. Other tools and methodologies related to high-level synthesis were also developed. We formulated a better algorithm to determine the minimum iteration period of any recursive DSP algorithm. We developed an exhaustive technique to locate all valid schedules and retimings of strongly connected data-flow graphs (DFGs), and we derived ILP models for efficient two-dimensional retiming. By extending the folding technique to include multirate constructs and developing a new approach to minimize the overall register usage, new and efficient architectures for discrete wavelet transforms using lattice-based architectures and tree-structured filter banks were developed. For digital subscriber loops, we investigated and characterized different approaches to minimizing the echo problem that are inherent with the transmission media. New efficient architectures for arithmetic operations within the finite field were developed and implemented. These new architectures were used to develop a fast and power efficient Reed-Solomon encoder. In our study of low-power design methodologies, we have developed a novel order-configurable architecture for FIR filters. A single chip can be configured as an FIR filter with a filter length up to 32 while consuming minimal power.

1 Introduction

The rapid design of high-performance and low-power dedicated digital signal processing (DSP) architectures requires appropriate selection of algorithm, architecture, and implementation style and usage of efficient synthesis tools. With the additional pressure of designing new high-speed architectures or re-designing an existing architecture that are area and power efficient in less time, the task becomes even more challenging. This is because to meet the new specifications, many new designs may need to be implemented using heterogeneous components where the algorithms and implementation styles used in the design are varied. For example at a lower level, there may exist functional units that implement full adders using a ripple carry or manchester carry algorithm, and within each alogorithm type there may exist adders that have implementation styles that are bit-serial, digit-serial, or bit-parallel. With these additional parameters and demands, the design space has become much larger and more uneven. Better tools and design methodologies become more important to quickly and efficiently search the space in an efficient manner.

Concurrent to developing tools for design space exploration, one must also investigate difficult DSP applications to understand and develop new design methodologies that can be extended into the developing CAD tools. By exploring the interaction between algorithm and architecture, one is able to gain a deeper understanding of the way different design tradeoffs and optimizations impact the final architecture.

In the past three years, we have addressed and developed CAD tools and design methodologies to perform high-level transformations and synthesis for DSP applications. On a parallel track, we have also investigated and developed design methodologies and architectures for discrete wavelet transforms, echo cancellers for high-speed digital subscriber loops, and finite field arithmetic for use in Reed-Solomon coders. Our goals were to develop fast and efficient techniques and tools that would reduce the time to explore the design space and locate an optimal design of application specific integrated circuits or ASICs for DSP applications. This report summarizes our approaches taken to achieve our goals, the algorithms that we utilized, and our experimental results that we gathered.

This report is divided into two main sections: CAD tools and architectures. Within the

CAD tools section we present the work performed in developing high-level synthesis tools (section 2) and other tools that we developed as we addressed the high-level synthesis problem (section 3). Under high-level synthesis, section 2.1 describes the Minnesota ARchitecture Synthesis (MARS) tool that is based on the loop-list heuristic approach and in section 2.2 we present our integer linear programming (ILP) models. In the other tools section, we present tools that solve problems which are related to the topic of high-level synthesis. We developed an algorithm that determines the minimum iteration bound of a data-flow graph (DFG) (section 3.1), a method in exhaustively locating all schedules and retimings for a given DFG (section 3.2), and a technique to perform two-dimensional retiming (section 3.3). In the architectures section, we present our work in developing algorithms and architectures that have high-performance, consume less power, and are area efficient for discrete wavelet transforms (section 4), echo cancellers for high-speed digital subscriber loops (section 5), finite field arithmetic for use in Reed-Solomon coders (section 6), and order-configurable FIR filters (section 7).

CAD Tools

2 High-Level Synthesis

In the past ten years, there has been a great deal of activity in developing high-level synthesis systems for automatic design of high performance, dedicated architectures, especially for digital signal processing (DSP) applications. Many of the more common techniques have been covered in tutorials and books [1] -[5]. More recent techniques include [6] -[26]. In designing real-time DSP systems, the use of high-level synthesis has become a more common and crucial step in the design flow because many real-time applications which require high sample rates or low power consumption can only be implemented by dedicated architectures. High-level synthesis can be viewed as a series of steps consisting of describing the behavior of the system to be designed as separate but interrelated operations (with either a high-level language or graph model such as a synchronous data-flow graph (DFG) [27]), selecting and allocating hardware resources, scheduling the operations to control time steps, and generating the control unit to synchronize the execution of the operations within the final design [1] [2]

[3]. Of the entire synthesis problem, hardware selection/allocation and scheduling are the two most difficult and crucial steps because decisions made here directly affect the final cost. Both of these tasks have been shown to be NP-complete [28]; therefore, many schedulers have been proposed with varying results and performance. Although heuristic methods can generate good results in short CPU time, they cannot guarantee optimal solutions. More formalized solutions using integer linear programming (ILP) techniques have been proposed [21]-[26] within the last few years. These models tend to be more flexible and are capable of generating optimal solutions but they suffer in exponential increases in run times as the model constraints become less restrictive.

Most of the previously developed synthesis systems assume that all same type operations will be assigned to one type of functional unit (or processor) (e.g., all addition operations will be processed by full adders). With this type of limited library, the solutions generated by these systems are not as cost optimal as solutions generated by systems using a library that contains multiple functional units for each type of operation. For example, Fig. 1 shows a simple DFG that consists of a set of identical nodes and are interconnected into two loops. Let us assume that the available library only contains one processor type, $P1$ which has a computational delay of 1 time unit (t.u.) and an area cost of 20 units. If the target iteration period for this DFG is 5 t.u., one possible processor allocation solution will require two $P1$ processors for a total area cost of 40 units and a valid final schedule is shown below:

time	1	2	3	4	5
$P1_1$	A	B	C	D	E
$P1_2$		F	G		

From this schedule, we can see that processor $P1_1$ is 100% utilized but processor $P1_2$ is only 40% untilized. If the processor library is expanded to include a second processor, $P2$ (with a computational delay of 2 t.u. and an area cost of 10 units), a better processor allocation can be generated. One solution will consist of one $P1$ and one $P2$ processor which will have a total area cost of 30 units and a valid final schedule is shown below:

time	1	2	3	4	5
$P1$	A	B	C	D	E
$P2$		F	F	G	G

This schedule shows that processor $P1$ is still 100% utilized and that processor $P2$ is 80% utilized. This solution also uses 25% less area than the previous solution.

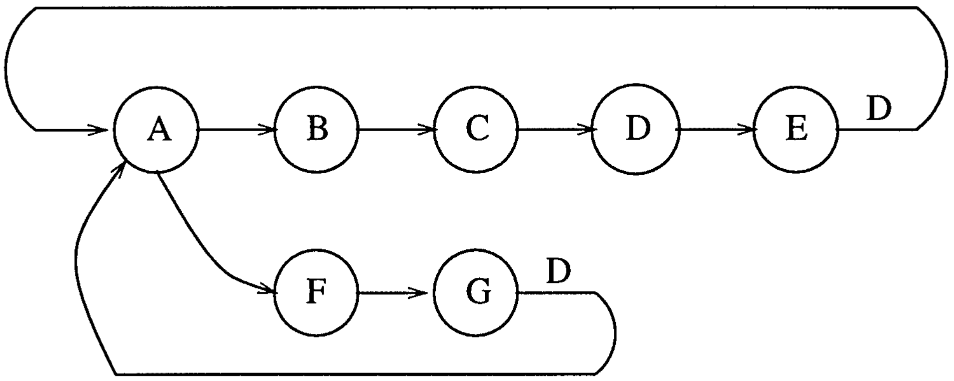


Figure 1: A simple DFG consisting of identical operations and two feedback loops.

More recently, a few systems allow for different types of processors for same type operations; however they only utilize homogeneous architectures where all functional units are implemented using a single implementation style such as bit-parallel [16] -[20], or bit-serial [6], [29]. Although this allows for expanded libraries and a slightly wider design space, it still restricts the design to one type of implementation style or word length.

We have developed two different solutions to high-level synthesis using heterogeneous functional unit libraries. One is based upon heuristic techniques which provide fast solutions but cannot guarantee their optimality and a second is based upon integer linear programming (ILP) models that can guarantee optimal solutions but suffer from exponential increases in run times as the design constraints are relaxed. In this section we provide more details of both techniques and provide results of our experiments using a small heterogeneous library.

2.1 MARS Design Tool

In our research we addressed the *automatic allocation of hardware functional units* from a heterogeneous library during the scheduling process to produce low cost area designs. In this tool, functional units include processors such as adders and multipliers and data format converters. The advantage of our approach is that we allow the design of heterogeneous architectures using different types of functional units (including implementation styles) to process same type operations. By utilizing a heterogeneous library, one removes the word size and implementation style restriction and allows the system to explore a much wider design space. However, if one allows the use of heterogeneous processors in the final architecture, the data format of one processor may not necessarily be the same as another processor. For example, the final design may contain an adder which computes one word in one clock cycle and a second adder which processes a half-word in one clock cycle. This leads to the need for data-format converters which accept input data in one format and generate output data in a different format (in our experiments, the data format may be bit-serial or digit-serial or bit-parallel). Therefore, the allocation, scheduling, and cost of these converters are also taken into account during the synthesis process. This high-level synthesis tool called the Minnesota ARchitecture Synthesis (MARS) System is based on our novel *iterative loop scheduling and allocation* technique that permits *implicit* retiming and pipelining. It also supports the unfolding transformation. In addition the synthesized architecture data-flow graph is generated by using the *folding* transformation.

2.1.1 MARS overview

The flowchart in Fig 2 displays the basic MARS framework. Our algorithm starts from the generation of the initial prototype schedule. The initial schedule helps the system generate a set of initial module solutions for the specified iteration period. The scheduling and resource refinement algorithm will then be invoked to determine the lowest cost processor and converter allocation that will produce a valid schedule for the given design parameters.

2.1.2 Loop-Based Synthesis

DSP algorithms are continuous and repetitive in nature; in other words, the operations are repeated in an iterative manner as new samples are processed. Because many DSP algorithms contain feedback (or recursive) loops , the operations of each loop for one iteration must be completed before the next iteration can be initiated and this imposes the greatest restrictions on the DFG. [30],[31]. Feedback limits the most obvious methods for improving the performance of the final architecture (e.g., pipelining). [31]. One cannot pipeline the feedback loops to any arbitrary level by inserting latches, because the pipelining latches would alter the number of delays in the loops and, hence, the original functionality of the

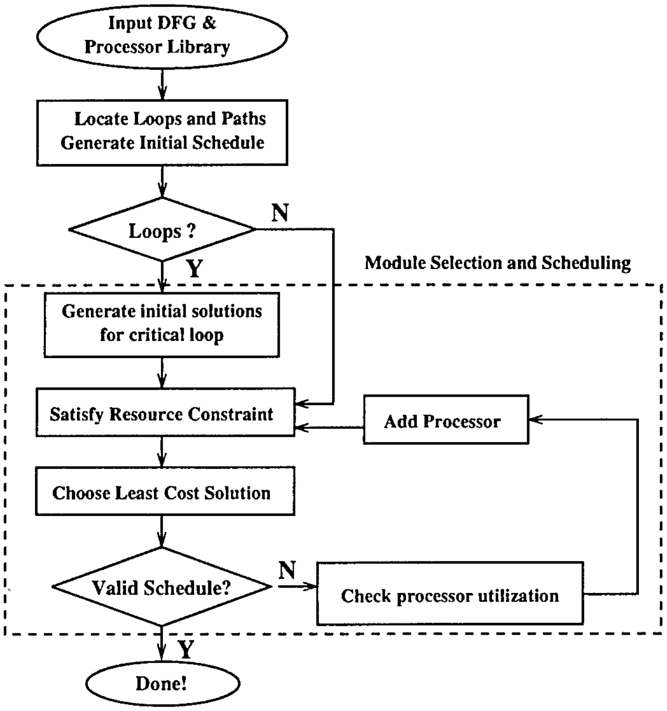

Input DFG &
Processor Library
Locate Loops and Paths
Generate Initial Schedule
Loops ?
N
Y
Module Selection and Scheduling
Generate initial solutions
for critical loop
Satisfy Resource Constraint
Add Processor
Choose Least Cost Solution
Valid Schedule?
N
Check processor utilization
Y
Done!

Figure 2: A flowchart showing the major steps of MARS-II.

DFG. The non-recursive (or feed-forward) sections are less restrictive because one can always pipeline these sections at the feed-forward cutsets to achieve the desired sample rate; but at the expense of greater latency. Because of this constraint, MARS first schedules the recursive operations followed by the non-recursive operations during the scheduling process. This methodology is known as the loop-based approach to high-level synthesis.

The first step of loop-based synthesis is to identify all of the loops. MARS utilizes the loop search algorithm described in [32] which has a complexity that is linear in the number of nodes plus edges. At this point, MARS also calculates the loop bound of every loop which will be used later in the synthesis process. The loop bound, T_{lb}, defines the minimum time required to complete one iteration of a loop, and is calculated as follows for loop j: $T_{lb_j} = T_{l_j}/D_{l_j}$, where T_{l_j} and D_{l_j} represent the computation time and the number of delays in loop j [31]. At first, MARS assumes that the operations are mapped to the fastest processors available in the library. This set of loop bounds define a lower bound on the iteration period, T, for the DFG. This bound, known as the iteration bound or T_∞, is the minimal time required for all recursive loops to complete one iteration and is determined by

Table 1: Library of Processor Types (wordlength=16)

type	processor	C	L	m	I/O
A_{bp}	Bit-parallel adder	1	1	53	bp
A_{hp}	Half-word parallel adder	1	2	19	hp
A_{ds}	4-bit digit-serial adder	1	4	6	ds
M_{bp}	Bit-parallel multiplier	5	1	331	bp
M_{hp}	Half-word parallel multiplier	6	2	173	hp
M_{ds}	4-bit digit-serial multiplier	9	5	86	ds

Table 2: Converter Types

type	conversion	C	L	m
$v_{bp,hp}$	$bp \rightarrow hp$	0	1	3
$v_{bp,ds}$	$bp \rightarrow ds$	0	3	4
$v_{hp,bp}$	$hp \rightarrow bp$	1	1	3
$v_{hp,ds}$	$hp \rightarrow ds$	0	2	3
$v_{ds,bp}$	$ds \rightarrow bp$	3	3	4
$v_{ds,hp}$	$ds \rightarrow hp$	2	2	3

locating the maximum loop bound [30],[31].

Because large number of loops may exist in the DFG, MARS reduces the set of all loops to a smaller subset which will be used for scheduling [14], [15]. If nonrecursive operations exist in the DFG, MARS locates all of the feed-forward paths which only contain nonrecursive operations. MARS also reduces this set of all paths [14], [15]. MARS now builds an initial schedule which will be used for generating initial solutions. Currently, MARS uses an 'As Soon As Possible' (ASAP) technique with the assumption that an infinite number of resources are available.

2.1.3 Module Selection

Module selection is the task of selecting a set of functional units from a processor library which is capable of satisfying all of the precedence constraints for the specified iteration period while minimizing a cost function. Table 1 shows the heterogeneous library used by MARS for our experiments. This library was also used in [25],[26] (see section 2.1.5 for comparison of results). This library includes both functional units and data-format converters. Each module description consists of the computational delay or latency, C, the pipeline period, L, the area cost, m, and its data format. The computational latency represents the time required for one operation to complete, from input to output (note that one computation does not necessarily compute a complete word). The pipeline period represents the minimum time required between successive word computations in the same functional unit.

The implementation of module selection within MARS is a two phase approach. In phase one, we generate a small number of initial solutions based on the characteristics of the loops and in phase two, we refine and generate a few more initial solutions based on the number of total operations in the DFG.

Every loop that has a loop bound equal to the iteration bound is considered a critical loop. Because critical loops are the most restrictive paths in the DFG, their operations must be scheduled onto the fastest processors of any solution. Therefore MARS utilizes the initial schedule to generate a few partial solutions to satisfy the critical loops. These solutions assume that all same type operations will be assigned to one processor (e.g., all additions to a bit-parallel adder). Any valid initial solution consisting of A_i type adders and M_j type multipliers must satisfy the following criteria for every critical loop, *cl*:

$$T \;\geq\; N_{M_{cl}} * ((C_{v_{i,j}} + C_{v_{j,i}}) + C_{M_j}) \;+\; N_{A_{cl}} * (C_{A_i}) \tag{1}$$

where T represents the iteration period, $N_{A_{cl}}$ and $N_{M_{cl}}$ represent the number of addition and multiplication operations in *cl*, C_{A_i} and C_{M_j} represent the computation time of the adder and multiplier, and $C_{v_{i,j}}$ and $C_{v_{j,i}}$ represent the pipeline latency of the data format conversion from i to j and from j to i, respectively.

Fig. 3 shows a solution generated by MARS for a simple IIR filter commonly known as a biquad filter. If we view the nodes as operations and ignore the data format converters, we can see that this filter contains two loops, of which one is critical. Using the fastest processors in the library shown in Table 1, the loop bounds are: $T_{lb_{L_1}} = 6$ t.u. and $T_{lb_{L_2}} = 3.5$ t.u. For an iteration period of 7 t.u., only two initial solutions satisfy equation 1: $S_1 = [A_{bp}, M_{bp}]$ and $S_2 = [A_{hp}, M_{hp}]$.

To measure the effectiveness of these initial solutions, we define a resource constraint inequality which when satisfied will ensure that there are enough time steps to which all operations may be scheduled if and only if no precedence constraints exist in the DFG:

$$\sum_{i=1}^{FU} \frac{PROC_{U_i}}{L_{U_i}} \geq \frac{N_U}{T} \tag{2}$$

where U and N_U represent an operation type and the total number of operations in the DFG, FU represents the number of functional unit types that can compute U, L_{U_i} is the pipeline period of the type i functional unit, and $PROC_{U_i}$ is a variable which represents the number of functional units of type i which can compute U.

Because the initial solutions were generated to satisfy the critical loops, they may not satisfy (2), especially for applications that contain nonrecursive operations. Note that (2) defines one linear inequality for each U type operation where the number of variables ($PROC_{U_i}$) are equal to the number of processors capable of computing a U type operation. If a solution to this inequality is also forced to satisfy:

$$min(\sum_{i=1}^{FU} PROC_{U_i} * m_{P_{U_i}}), \tag{3}$$

we can use (2) and (3) to refine the initial solutions or generate new ones (min() is a minimizing function).

In the biquad filter example, one of the initial solutions did not satisfy (2). Therefore MARS has to refine the set of initial solutions. The refined initial solutions are: $S_1 = [A_{bp}, M_{bp}]$ (cost = 384), $S_2 = [A_{hp}, A_{ds}, M_{hp}, M_{ds}]$ (cost = 284).

2.1.4 Scheduling

For scheduling and resource allocation, we use an iterative approach that includes an *incremental allocation and elimination* refinement step to achieve the low cost solution. After MARS generates a set of initial solutions, it chooses the low cost solution to become the solution-under-test (SUT). The final step is to use the SUT (with allocated data format converters) and verify if a valid schedule can be constructed. The MARS scheduler will start from the initial schedule and then steps through the schedule at each time step; bind operations to processors. During the scheduling, some time steps may contain resource conflicts (when more operations are scheduled at a time step than available processors). To resolve a conflict at a time step, MARS uses a simple priority function which identifies an operation to be bound to a processor at that time step (additional data format converters, if needed, are also allocated at this time). After all available processors at a time step are exhausted, the remaining unbound operations are reassigned to the next time step. This technique is repeated until a valid schedule is obtained or MARS encounters a time step where the resource conflicts cannot be resolved.

Currently the priority values are based upon two criteria: the flexibility available to an operation, and the type of successor operation. We define flexibility, F, to be the number of time steps in which an operation can be assigned. The flexibility for operations in the loops can be easily calculated before performing the scheduling step:

$$F_{l_i} = T * D_{l_i} - T_{lb_i}$$

where F_{l_i} is the flexibility associated with loop l_i (note that T_{lb_i} is determined by using the fastest processors in the SUT). Each loop will have its own flexibility and operations which belong to multiple loops will have a flexibility equal to the smaller loop flexibility. Non-recursive operations have infinite flexibility because we do not place restrictions on the latest execution time. Operations that have greater flexibility will have lower priority and operations in the critical loops will have the highest priority. As the scheduling process progresses, the flexibilities change as operations are bound to the processors of the SUT or reassigned to new time steps. The flexibilities are also affected by the allocation of required data-format converters.

The second criterion to determine the priority value is only considered if two or more operations have the same flexibility. This criterion checks the operation type of the successor of each operation that is being considered for processor binding or time step reassignment. MARS gives higher priority to operations that have successor operations which provide for greater overlap of different operation types between the loops.

For cases where MARS cannot resolve all resource conflicts, the SUT becomes invalid. Instead of eliminating the SUT, MARS uses an iterative approach that allows for *incremental processor refinement* for invalid solutions. Let us assume that the current invalid SUT contains 1 A_{bp} and 2 A_{hp}. MARS will check the utilization of each functional unit in the SUT and then make an incremental refinement on the SUT. If all processors have an operation bound to it, MARS allocates another lowest cost functional unit of the SUT (e.g., another A_{hp}). However, if one of the A_{hp} is never utilized, MARS would not allocate another A_{hp}. Instead, MARS would remove one A_{hp} and allocate one A_{bp} unit. This simple refinement step allows MARS to avoid unnecessary allocation of functional units that will never be used. The cost of this newly refined solution is then compared with the other initial solutions generated earlier and the lowest cost solution becomes the new SUT. This iterative loop continues until a valid schedule can be generated for a SUT which becomes the low cost solution.

For the biquad filter example, S_2 is the initial low cost solution and it becomes the SUT. MARS is able to produce a valid schedule for S_2 as shown with the DFG of the final solution (including data-format converters) in Fig. 3.

2.1.5 Experimental Results

The fifth-order wave digital elliptic filter has been used extensively for high-level synthesis [1]; therefore, we ran a series of experiments using various libraries found in previous work related to module selection. Table 3 contains the results of module selection for a small library presented in [19]. This library only contains non-pipelined processors consisting of two adders (A_{fast} with a computation time of 1 t.u., and A_{slow} with a computation time of 2 t.u.) and one multiplier (M with a computation time of 2 t.u.). In Table 3 the first column shows the results of [19] and the second column contains the results produced by MSSR [20] (note that the experiments with MSSR were performed with only A_{fast} and M processors).

time	1	2	3	4	5	6	7
A_{hp}:	2	2	1	1	4	4	
A_{ds}:		3	3	3	3		
M_{hp}:		5	5	6	6	7	7
M_{ds}:	8	8	8	8			

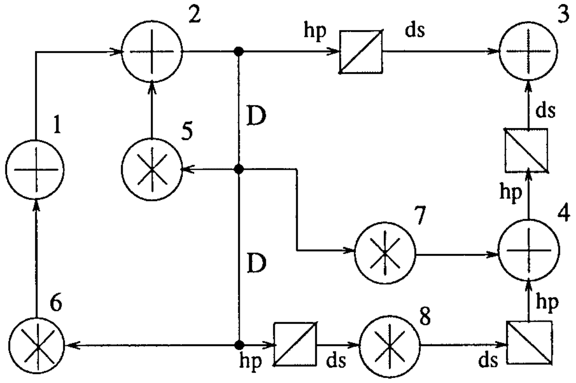


Figure 3: The valid schedule and DFG of the biquad filter showing the final assignment of operations to processor types and the data-format converters.

We also show the optimal solutions generated by the ILP models of [25] along with the results of MARS. This table shows that MARS is able to generate optimal solutions while [19] and MSSR could not for all cases.

Table 3: Comparison with [10], MSSR, ILP models, and MARS using the homogeneous library from [10].($C_{A_{fast}} = 1, C_{A_{slow}} = 2, C_M = 2$)

	[19]	MSSR [20]	ILP	MARS
16	NA	NA	$3A_{fast}$, $2M$	$3A_{fast}$, $2M$
17	$3A_{fast}$, $3M$	$3A_{fast}$, $3M$	$2A_{fast}$, $2M$	$2A_{fast}$, $2M$
18	$2A_{fast}$, $2M$	$2A_{fast}$, $2M$	$2A_{fast}$, $2M$	$2A_{fast}$, $2M$
20	NA	NA	$1A_{fast}$, $1A_{slow}$, $1M$	$1A_{fast}$, $1A_{slow}$, $1M$
21	$1A_{fast}$, $1A_{slow}$, $1M$	$2A_{fast}$, $1M$	$1A_{fast}$, $1A_{slow}$, $1M$	$1A_{fast}$, $1A_{slow}$, $1M$
26	NA	NA	$3A_{slow}$, $1M$	$3A_{slow}$, $1M$
28	$1A_{fast}$, $1M$	NA	$1A_{fast}$, $1M$	$1A_{fast}$, $1M$
54	$1A_{slow}$, $1M$	NA	$1A_{slow}$, $1M$	$1A_{slow}$, $1M$

We also ran experiments on a larger library of non-pipelined processors used by MSSR [20] and these results are shown in Table 4. We compare our results with those of MSSR and of the ILP models of [25] in Table 5. Here we also show the cost of the solution and the run times in CPU seconds as run on a DECstation 3100 with 16MB of memory (Note: the CPU times for MSSR is an average time over all examples). The ILP models were solved on a SUN Sparcstation 20 and the models became too large to solve for $T \geq 65$. From Table 5 we can see that MARS generates better solutions than MSSR and in less time. Although ILP models can provide optimal solutions, this table also shows that they can become too large to solve.

Table 4: Homogeneous library used by MSSR [11] (non-pipelined processors, C = L).

Add	C	L	m	Mult	C	L	m
A_1	1	1	16	M_1	1	1	256
A_2	4	4	5	M_2	16	16	32
A_3	16	16	2	M_3	256	256	2

We also experimented with several other common DSP benchmarks using the heterogeneous processor and converter library shown in Table 1. In Table 6 we directly compare the performance of MARS with the ILP models developed for the same problem (note, the experiments for both were performed on a SUN Sparcstation 2 unless marked by an '*' which were run on a SUN Sparcstation 20). This table has been broken down into three sections,

Table 5: Comparison of MSSR, ILP models, and MARS using the homogeneous library used by MSSR.

	MSSR [20]			ILP^*		MARS		
T	Allocation	Cost	CPU	Allocation	Cost	Allocation	Cost	CPU
13	NA	-	-	$3A_1$, $1M_1$	304	$3A_1$, $1M_1$	304	0.35
14	$4A_1$, $2M_1$	576	9.5*	$3A_1$, $1M_1$	304	$3A_1$, $1M_1$	304	0.35
15	$3A_1$, $1M_1$	304	9.5*	$2A_1$, $1M_1$	288	$2A_1$, $1M_1$	288	0.32
16	$2A_1$, $1A_2$, $1M_1$	293	9.5*	$2A_1$, $1M_1$	288	$2A_1$, $1M_1$	288	0.32
18	$2A_1$, $1M_1$	288	9.5*	$2A_1$, $1M_1$	288	$2A_1$, $1M_1$	288	0.32
22	NA	-	-	$1A_1$, $1A_2$, $1A_3$, $1M_1$	279	$1A_1$, $2A_2$, $1M_1$	282	0.40
23	NA	-	-	$1A_1$, $1A_2$, $1M_1$	277	$1A_1$, $1A_2$, $1M_1$	277	0.32
27	$1A_1$, $1M_1$	272	9.5*	$1A_1$, $1M_1$	272	$1A_1$, $1M_1$	272	0.33
58	NA	-	-	$3A_1$, $4M_2$	176	$3A_1$, $4M_2$	176	4.20
60	$3A_1$, $4M_2$	176	9.5*	$2A_1$, $4M_2$	160	$2A_1$, $4M_2$	160	3.60
64	NA	-	-	$1A_1$, $4M_2$	144	$1A_1$, $4M_2$	144	3.60
70	$1A_1$, $4M_2$	144	9.5*	Too Large	-	$1A_1$, $4M_2$	144	3.60
74	NA	-	-	Too Large	-	$1A_1$, $2M_2$	80	0.86
93	$1A_1$, $2M_2$	80	9.5*	Too Large	-	$1A_1$, $2M_2$	80	0.86
140	NA	-	-	Too Large	-	$1A_2$, $1M_2$	37	0.50
156	$1A_2$, $1M_2$	37	9.5*	Too Large	-	$1A_2$, $1M_2$	37	0.50
240	NA	-	-	Too Large	-	$2A_3$, $1M_2$	36	0.72
288	$2A_3$, $1M_2$	36	9.5*	Too Large	-	$2A_3$, $1M_2$	36	0.72
432	NA	-	-	Too Large	-	$1A_3$, $1M_2$	34	1.10
448	$1A_3$, $1M_2$	34	9.5*	Too Large	-	$1A_3$, $1M_2$	34	1.10
944	NA	-	-	Too Large	-	$2A_3$, $4M_3$	12	9.30
1040	$2A_3$, $4M_3$	12	9.5*	Too Large	-	$1A_3$, $4M_3$	10	11.40

the desired iteration period, the results and performance of MARS, and the results and performance of the ILP models of [25].

In results columns, Table 6 shows the final solution (processor and converter allocation), the final cost, and the CPU time in seconds. Table 6 shows that MARS can generate similar solutions in one to two orders of magnitude less time than the ILP approach (See section 2.2).

Table 6: Time assignment benchmark results and comparisons between MARS and ILP models for the heterogeneous library of Table 1.

T	MARS	cost	CPU	ILP model	cost	CPU
5th Order Wave Elliptic Filter						
25	$3A_{bp},1M_{bp}$	490	0.36	$3A_{bp},1M_{bp}$	490	3.16
26	$2A_{bp},1M_{bp}$	437	0.32	$2A_{bp},1M_{bp}$	437	26.2
27	$1A_{bp},2A_{hp},1M_{bp},2v_{bp,hp},1v_{hp,bp}$	431	0.23	$1A_{bp},2A_{hp},1M_{bp},1v_{bp,hp},1v_{hp,bp}$	428	658
28	$1A_{bp},1A_{hp},1M_{bp},1v_{bp,hp},1v_{hp,bp}$	409	0.88	$1A_{bp},1A_{hp},1M_{bp},1v_{bp,hp},1v_{hp,bp}$	409	417
31	$3A_{hp},1M_{hp}$	230	0.37	$3A_{hp},1M_{hp}$	230	–*
34	$2A_{hp},,1A_{ds},1M_{hp}$	223	0.39	$2A_{hp},1M_{hp}$	211	–*
4th Order Lattice Filter						
14	$3A_{bp},2M_{bp}$	821	0.20	$3A_{bp},2M_{bp}$	821	1.58
15	$2A_{bp},1M_{bp}$	437	0.20	$2A_{bp},1M_{bp}$	437	3.15
16	$1A_{bp},2A_{hp},1M_{bp}, 1v_{bp,hp},1v_{hp,bp}$	428	0.30	$1A_{bp},1A_{hp},1M_{bp},1v_{bp,hp},1v_{hp,bp}$	409	18.0
17	$1A_{bp},1A_{hp},1M_{bp},1v_{bp,hp},1v_{hp,bp}$	409	0.48	$1A_{bp},1M_{bp}$	384	21.2
18	$4A_{hp},1M_{hp}$	249	0.20	$2A_{hp},1A_{ds},1M_{hp},1v_{hp,ds},1v_{ds,hp}$	223	11.9
4th Order Jaumann Filter						
16	$2A_{bp},1M_{bp}$	437	0.15	$2A_{bp},1M_{bp}$	437	14.9
17	$1A_{bp},1M_{bp}$	384	0.17	$1A_{bp},1M_{bp}$	384	14.3
18	$1A_{bp},1M_{bp}$	384	0.73	$1A_{bp},1M_{bp}$	384	39.9
19	$3A_{hp},1M_{hp}$	230	0.18	$2A_{hp},1M_{hp}$	211	24.7
20	$2A_{hp},1M_{hp}$	211	0.15	$2A_{hp},1M_{hp}$	211	48.6
23	$4A_{ds},1M_{hp}, 1v_{hp,ds},1v_{ds,hp}$	203	0.17	-	-	-
24	$3A_{ds},1M_{hp}, 1v_{hp,ds},1v_{ds,hp}$	197	0.43	-	-	-
4 stage Pipelined Lattice Filter						
3	$2A_{bp},7A_{ds},5M_{bp}, 6v_{bp,ds}$	1827	0.27	$2A_{bp},7A_{ds},5M_{bp}, 6v_{bp,ds}$	1827	23.6
4	$1A_{hp},9A_{ds},3M_{bp},1M_{hp},2M_{ds},1v_{bp,hp}$ $1v_{hp,ds},1v_{hp,bp},7v_{bp,ds},2v_{ds,hp},1v_{ds,bp}$	1458	0.30	$2A_{hp},7A_{ds},4M_{bp},$ $2v_{bp,hp},6v_{bp,ds}, 1v_{hp,bp}, 1v_{hp,ds}$	1440	58.2
5	$11A_{ds},3M_{bp}, 9v_{bp,ds}, 3v_{ds,bp}$	1107	0.72	$9A_{ds},3M_{bp}, 9v_{bp,ds}, 2v_{ds,bp}$	1091	40.6
6	$9A_{ds},2M_{bp},1M_{hp},$ $1v_{hp,ds},2v_{ds,bp},6v_{bp,ds}, 1v_{ds,hp}$	927	0.65	$8A_{ds},2M_{bp},$ $6v_{bp,ds},1v_{hp,ds}, 1v_{ds,bp},1v_{ds,hp}$	917	77.2
16 Point FIR Filter						
1	$60A_{ds},8M_{bp}, 24v_{bp,ds},24v_{ds,bp}$	3200	0.30	$60A_{ds},8M_{bp}, 24v_{bp,ds},24v_{ds,bp}$	3200	3.53
2	$30A_{ds},4M_{bp}, 12v_{bp,ds},16v_{ds,bp}$	1616	0.25	$30A_{ds},4M_{bp}, 12v_{bp,ds},12v_{ds,bp}$	1600	5.65
3	$26A_{ds},3M_{bp},8v_{bp,ds},8v_{ds,bp}$	1213	0.95	$20A_{ds},3M_{bp},8v_{bp,ds},8v_{ds,bp}$	1177	7.85
4	$15A_{ds},2M_{bp}, 6v_{bp,ds},8v_{ds,bp}$	808	0.28	$15A_{ds},2M_{bp}, 6v_{bp,ds},6v_{ds,bp}$	800	7.25
5	$15A_{ds},1M_{bp},1M_{hp},1M_{ds},$ $1v_{hp,ds},3v_{bp,ds},2v_{ds,hp},5v_{ds,bp}$	721	0.49	$12A_{ds},1M_{bp},3M_{ds},3v_{ds,bp},3v_{bp,ds}$	685	20.4
6	$13A_{ds},1M_{bp},2M_{ds},3v_{bp,ds},6v_{ds,bp}$	617	0.64	$10A_{ds},1M_{bp},1M_{hp},$ $1v_{bp,ds},1v_{hp,ds},1v_{ds,bp},1v_{ds,hp}$	578	121.87*
7	$12A_{ds},1M_{bp},1M_{ds},3v_{bp,ds},7v_{ds,bp}$	529	0.55	-	-	-
8	$8A_{ds},1M_{bp},3v_{bp,ds},8v_{ds,bp}$	423	0.30	-	-	-

2.2 Integer Linear Programming High-Level Synthesis

As stated earlier, integer linear programming (ILP) solutions have been recently used to solve the scheduling problem in high-level synthesis. By modeling the scheduling task as an ILP problem, the models provide the flexibility to include new design considerations and optimal solutions. Therefore the ILP formulation is ideal for modeling the scheduling task in a heterogeneous synthesis environment. In our research, we have developed a set of efficient ILP models for high-level DSP synthesis within a heterogeneous environment. This approach leads to faster solutions than other ILP approaches by bounding the search space of the variables. Furthermore, this approach can also perform automatic retiming and pipelining as well as unfolding to improve the processor utilization. These models have been designed to perform *automatic allocation of hardware functional units* from a heterogeneous library during the scheduling process while minimizing the overall area cost. The functional units in these models include processors, data format converters, and registers.

2.2.1 Time-Constrained Scheduling by ILP

The time-constrained scheduling determines when and in which processor each computation should be executed to minimize the cost, such as the number of processors, while satisfying the speed requirement. The time assignment step determines the execution time of each node in the data-flow graph (DFG). It is followed by the processor allocation step which determines in which processor each computation is executed. In this section, the integer linear programming model for time assignment supporting overlapped schedule (or functional pipelining) and structural pipelining is introduced.

We use the following notation to describe a synchronous DFG. DFG $= (N, E)$ where N is the set of nodes and E is the set of edges in the DFG. Each node $i \in N$ has a scheduling range defined by a lower and upper bound, LB_i and UB_i. These are the earliest and the latest time steps, respectively, in the scheduling range. LB_i and UB_i can be determined as the *as soon as possible* (ASAP) schedule and the *as late as possible* (ALAP) schedule, respectively. R_i denotes the scheduling range of node i, which is the closed time interval $[LB_i, UB_i]$. We define $R_i + k$ to denote the interval $[LB_i + k, UB_i + k]$ where k is an any integer.

Let C_a and L_a denote the *computation latency* and the *pipeline period* of node a, respectively. The computation latency represents the time from an input to its associated output. If the computation of node a starts at time step j, the result is output at time step $j + C_a$. The pipeline period represents the minimum time between successive computations. If the computation of node a is initiated at time step j on a processor, any other computation cannot be initiated on the same processor until $j + L_a$.

The ILP model minimizes the the cost, M, which is the number of processors (4) (in the case when only one type of processor is used), subject to the constraints (5), (6), and (7). The following parameters are used in the ILP model.

T_r is the specified iteration·period.

$i \in N$ is a node.

j is a time step.

$x_{i,j}$ is a binary variable, and $x_{i,j} = 1$ means that the computation of the node i starts at the time step j.

$e = (a, b) \in E$ is an edge directed from node a to node b with a delay count W_e.

C_a is the computation latency of node a.

L_a is the pipeline period of node a.

$$\text{Minimize } COST = M \tag{4}$$

$$\sum_{j \in R_i} x_{i,j} = 1 \quad \forall i \in N \tag{5}$$

$$\sum_{ja=j-C_a+1}^{UB_a} x_{a,ja} + \sum_{jb=LB_b}^{j-W_eT_r} x_{b,jb} \leq 1 \quad \forall e = (a, b) \in E, j \in (R_a + C_a - 1) \cap (R_b + W_eT_r), \tag{6}$$

$$\sum_{i \in N} \left\{ \sum_{k_1=\lfloor \frac{LB_i}{T_r} \rfloor}^{\lceil \frac{UB_i}{T_r} \rceil} \sum_{p=0}^{L_i - \lfloor \frac{L_i-1}{T_r} \rfloor T_r - 1} x_{i,J+k_1T_r-p} + \left\lfloor \frac{L_i - 1}{T_r} \right\rfloor \right\} \leq M \quad J = 0, 1, \ldots, T_r - 1 \tag{7}$$

The assignment constraint (5) ensures that each node i has only one start time in its scheduling range R_i.

For every directed edge (a, b), the computation of node b must start after the computation of node a is completed. This is ensured by the precedence constraint (6).

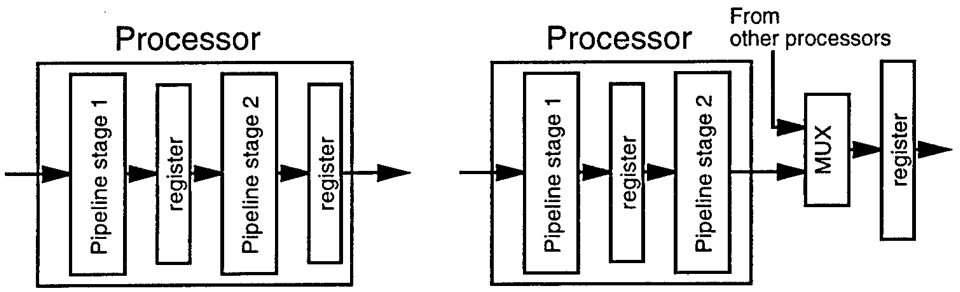
Given the iteration period T_r, the time class $J = 0, 1, \ldots, T_r - 1$ must hold. Each time step j belongs to the time class $J = j - \lfloor \frac{j}{T_r} \rfloor T_r$. In other words, the time class J consists of time steps J, $J + T_r$, $J + 2T_r, \ldots$ In the overlapped schedule, the computations executed at time steps belonging to the same time class are executed concurrently in different processors. The inequality (7) is used to count the required number of processors. The first term of the left-hand side of (7) is the number of nodes whose computation is initiated or being executed at the time class J. When the pipeline period of a node is longer than the iteration period, the processor must be counted multiple times, $\left\lfloor \frac{L_a - 1}{T_r} \right\rfloor$, since the node occupies the processor for more than one iteration period. This accounts for the second term in constraint (7). The inequalities (7) for all the time classes make the integer variable M no less than the largest required number of processors.

2.2.2 Counting the Number of Registers During Scheduling

In [33], a technique was proposed to count the number of registers during resource-constrained scheduling. Since the technique was developed for non-overlapped scheduling, it cannot be directly applied to the time-constrained overlapped scheduling. In this section, we generalize the technique for the overlapped scheduling. Furthermore, it is extended so that registers of general digit-serial data can be counted.

2.2.2.1 The Models of Processors and Registers

The computation latency is the difference in time steps from an input of a data to an output of a result associated with that input data. Let C_a denote the computation latency of a processor executing the computation of node a. If the computation of node a starts at time step j, its result becomes available as input data to computations of other nodes at time step $j + C_a$. Here, 'available' means the data is stored in a register and can be read by processors after and on the time step $j + C_a$. From this view point, there are two models of processors: one where a processor has its own dedicated register to store the output data as illustrated in Fig.4(a); and the other where a processor does not have such a register at the output as illustrated in Fig.4(b). In the latter case, the computed result is latched by a register outside the processor at the end of the time step $j + C_a - 1$ and the data becomes



(a) (b)

Figure 4: Processor models. In this case, processors are pipelined in two levels. (a) A processor has its own output register. (b) A processor does not have its own output register.

available from the time step $j+C_a$. Registers which are not dedicated to particular processors can be shared or commonly used by all processors (note that processors may have their own internal registers for pipelining but these registers cannot be commonly used by processors). Although the latter model would impose longer logic level critical path on the last pipeline stage of a processor, it could lead to synthesized systems which use less number of registers and therefore less chip area. In this paper, we use the model of Fig.4(b). Processors are assumed to have no dedicated registers at the output.

2.2.2.2 The Technique to Count the Number of Registers

In this section, the technique to count the number of registers proposed in [33] is briefly introduced.

The *life-time* of data is defined as the duration from the time step the data is produced to the time step the data is last used. If the life-time of data contains a time step j, the data is said to be *live* at j. The data live at time step j must be stored in a register at j. Therefore, the required number of registers at a particular time step is equal to the number of data live at that time step. Let b_a denote the node which last uses the data output from node a. Note that the output data of node a becomes live at the time step j if the computation of node a begins at the time step $j - C_a$. Whether the data produced by the execution of node a is live at time step j is checked by

$$\sum_{ja=LB_a}^{j-C_a} x_{a,ja} - \sum_{ja=j-C_a+1}^{UB_a} x_{a,ja} - \sum_{jb=LB_b}^{j-1} x_{b_b,jb} + \sum_{jb=j}^{UB_b} x_{b_b,jb} = \begin{cases} 2 & \text{if data is live at } j \\ 0 & \text{if data is not live at } j \end{cases} . \quad (8)$$

By summing the left-hand side of (8) for all the nodes in N, we get twice the number of live data at time step j. Thus, we obtain

$$\sum_{a\in N}\left(\sum_{ja=LB_a}^{j-C_a} x_{a,ja} - \sum_{ja=j-C_a+1}^{UB_a} x_{a,ja} - \sum_{jb=LB_b}^{j-1} x_{b_b,jb} + \sum_{jb=j}^{UB_b} x_{b_b,jb}\right) \leq 2M_R \tag{9}$$

where M_R is the number of registers. By applying the inequality (9) to every time step j, the required number of registers, M_R, can be obtained.

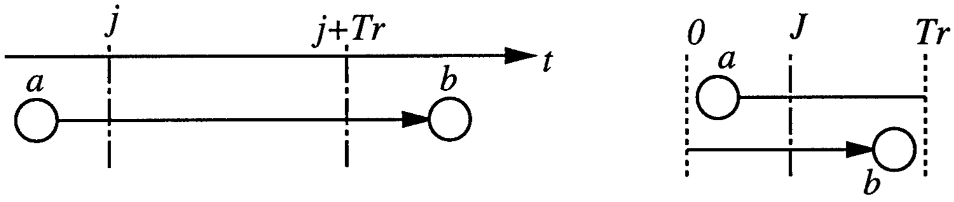
In general, the node a may have more than one immediate successor nodes. If i is an immediate successor node of node a, then the edge (a, i) must exist in E. The life-time of the data output by node a ends at the time step when the last immediate successor node is executed. Generally, we cannot know prior to scheduling which immediate successor node is executed last. Therefore, we must use the inequalities (9) for all edges if which successor node is last executed is not known [34]. This is represented as

$$\sum_{(a,b)\in E_0}\left\{\sum_{ja=LB_a}^{j-C_a} x_{a,ja} - \sum_{ja=j-C_a+1}^{UB_a} x_{a,ja} - \sum_{jb=LB_b}^{j-1} x_{b,jb} + \sum_{jb=j}^{UB_b} x_{b,jb}\right\} \leq 2M_R$$
$$\forall E_0 \in E_s,\ J = 0, 1, \ldots, T_r - 1. \tag{10}$$

where E_s is the set of edge sets where each element set, E_0, is the set of edges such that no two edges have the same starting node. Thus, each E_0 corresponds to a combination of nodes and the immediate successor nodes. Theoretically, there exist $\prod_{a\in N} s_a$ combinations of such edges and therefore $\prod_{a\in N} s_a$ elements in E_s, where s_a is the number of immediate successor nodes of node a. However, whether an immediate successor node would last use the data may be known prior to scheduling by means of transitivity analysis. We can reduce E_s by eliminating some element sets E_0 which contain the edge (a, b) where node b is known not to be the last node to use the data of node a.

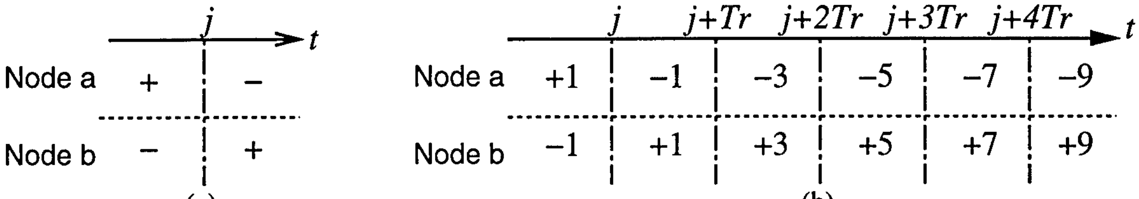
2.2.2.3 The Number of Registers in Overlapped Schedule

While non-overlapped scheduling of an iterative processing algorithm derives the schedule where all the computations in the current iteration are executed within an iteration period, the computations in the current iteration are distributed over several iteration periods in overlapped schedules [35, 36, 37]. Therefore, execution of current iteration overlaps with the previous and subsequent iterations. In this case, the life-time of a data may be longer than the iteration period and may overlap with itself for some time classes as shown in Fig.5. We



(a) (b)

Figure 5: Register usage in an overlapped schedule. (a) The life-time is longer than the iteration period T_r. Then, two registers are used at the time class J as shown in (b).



(a) (b)

Figure 6: Dividing time steps into groups and assignment of coefficients. (a) For nonoverlapped scheduling. (b) For overlapped scheduling.

must use as many registers as the number of overlaps for storing any data. Therefore, to count the number of registers precisely, the life-time of data must be checked to examine not only whether it contains a particular time class but also how many times it contains that time class.

In the technique to count the number of live data in [38, 33], time steps are divided at time step j into two domains. For node a, the variables x_{a,j_a} are accumulated into (9) with a coefficient +1 for $j_a \le j - C_a$ and −1 for $j_a > j - C_a$. For the immediate successor node b_a, the variables x_{b_a,j_b} are accumulated into (9) with a coefficient −1 for $j_b < j$ and +1 for $j_b \ge j$. This is illustrated in Fig.6(a).

Time steps can also be divided into more than two domains if necessary. We divide time steps at time class J, that is the time steps $J + kT_r$ for integers k, as illustrated in Fig.6(b). Then, we associate the coefficient $1-2k$ to the variables x_{a,j_a} where $J+(k-1)T_r-C_a < j_a \le J+kT_r-C_a$ and the coefficient $-1+2k$ to the variables x_{b_a,j_b} where $J+(k-1)T_r \le j_b < J+kT_r$ and accumulate the variables to derive the following inequality

$$\sum_{(a,b)\in E_0} \sum_{k=\underline{k_{ab}}}^{\overline{k_{ab}}} \left(\sum_{ja=J+(k-1)T_r-C_a+1}^{J+kT_r-C_a} (-2k+1)x_{a,ja} + \sum_{jb=J+(k-1)T_r}^{J+kT_r-1} (2k-1)x_{b,jb} \right) \le 2M_R, \tag{11}$$

where the upper and lower bound of k, $\underline{k_{ab}}$ and $\overline{k_{ab}}$, respectively, are chosen so that every time step in the scheduling ranges R_a and R_b for the edge (a, b) is included. They are calculated as

$$\begin{aligned} \underline{k_{ab}} &= \min\left\{ \left\lceil \frac{LB_a + C_a + J}{T_r} \right\rceil, \left\lceil \frac{LB_b + 1 - T_r + 1 - J}{T_r} \right\rceil \right\}, && (12) \\ \overline{k_{ab}} &= \max\left\{ \left\lceil \frac{UB_a + C_a + T_r - 1 - J}{T_r} \right\rceil, \left\lceil \frac{UB_b + 1 - J}{T_r} \right\rceil \right\}. && (13) \end{aligned}$$

For example, if the node a and its immediate successor node b are scheduled at the time step between $J - C_a$ and $J + T_r - C_a$ and the time step between $J + 3T_r$ and $J + 4T_r$, respectively, the life-time of the data output by node a contains the time class J three times, i.e., the time steps $J + T_r$, $J + 2T_r$, and $J + 3T_r$. In this case, the left-hand side of (11) becomes 6, that is -1 for node a and $+7$ for node b. Therefore, the left-hand side of (11) gives exactly twice the times the life-time contains the time class J.

Moreover in the overlapped scheduling, the number of delays on the edges are considered in the precedence constraints. The number of registers depends on the number of delays. If the number of delays on the edge $e = (a, b)$ is W_e, then the data output by the computation of node a is used by the node b after W_e iterations. In other words, the life-time of the data contains a particular time class another W_e number of times. Therefore, the inequality (11) is modified by taking the delays into account as follows:

$$\sum_{(a,b)\in E_0} \sum_{k=\underline{k_{ab}}}^{\overline{k_{ab}}} \left(\sum_{ja=J+(k-1)T_r-C_a+1}^{J+kT_r-C_a} (-2k+1)x_{a,ja} + \sum_{jb=J+(k-1)T_r}^{J+kT_r-1} (2(k+W_e)-1)x_{b,jb} \right) \leq 2M_R$$
$$\forall E_0 \in E_s,\ J = 0, 1, \ldots, T_r - 1. \quad (14)$$

The ILP model for the time assignment to minimize the total cost of processors and registers is as follows. It minimizes the cost (15), subject to the constraints (5), (6), (7), and (14).

$$\text{Minimize } COST = mM + m_r M_R \quad (15)$$

where m and m_r are the relative costs of a processor and a register.

2.2.2.4 Registers for Digit-Serial Data Architecture

Digit-serial architecture is used where the inexpensive bit-serial architecture is too slow and the expensive bit-parallel architecture is faster than necessary [39, 40]. The number of bits processed per cycle is referred to as the digit-size. The bit-serial architecture and the bit-parallel architecture can be regarded as special cases of the digit-serial architectures for digit-size equal to 1 and the word-length, respectively. In this section, the technique to count the number of registers is extended for digit-serial architectures.

If the word-length is w bits and the digit-size is d bits, then one word of data consists of w/d digits. We consider the case where w is a multiple of d. Let n denote the number of digits in a word, i.e., $n = w/d$. If the first digit of a data is input to node a at time step j_0, then the second digit is input at $j_0 + 1$, and the last digit is input at $j_0 + n - 1$. The computation latency C_a of node a is the time difference from the input of i-th digit to the output of i-th digit for $i = 0, 1, \ldots, n-1$. Hence, in the case mentioned above, the first digit of the output is available at time step $j_0 + C_a$, the second digit is available at $j_0 + C_a + 1$, and the last digit is available at $j_0 + C_a + n - 1$.

A digit-serial register is the set of d 1-bit registers. One digit-serial register stores one digit at a time. While storage of a bit-parallel data always requires one bit-parallel register, the required number of digit-serial registers for storing digit-serial data varies from time step to time step even in the non-overlapped schedule. Fig.7 shows the schedules of node a and its immediate successor node b. We assume that $n = 4$, $C_a = 2$. The arrow in the figure represents the life-time of a digit. For example, the life-time of the first digit of the

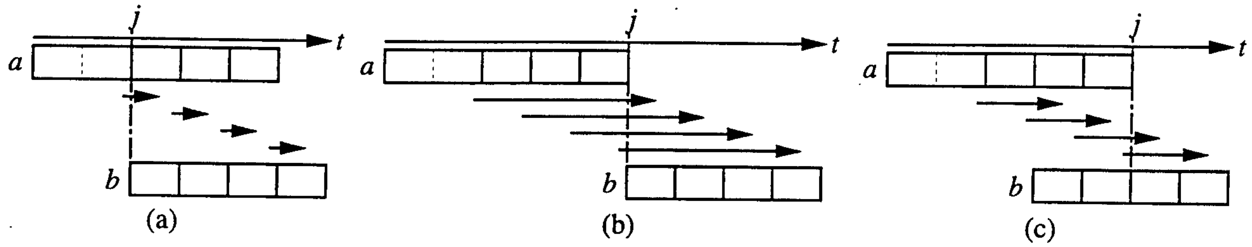


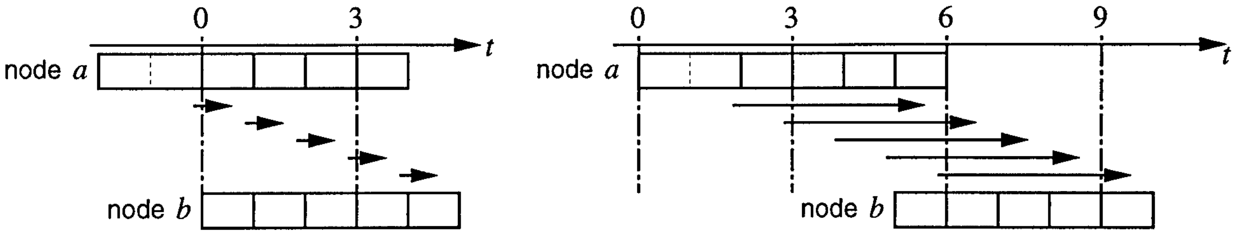
Figure 7: The number of live digits ($n = 4$, $C_a = 2$).

data output by node a contains only the time step j in Fig.7(a) while the life-time of the first digit of the data output by node a contains the time steps $j-3, j-2, j-1$, and j in Fig.7(b). In the schedule shown in Fig.7(a), only the life-time of the first digit contains the time step j. Therefore, we need only one digit-serial register in this case. In the schedules shown in Fig.7(b) and (c), the time step j are contained by all the digits and the 3rd and 4th digits, respectively. Therefore, we need 4 and 2 digit-serial registers in the schedules shown in Fig.7(b) and (c), respectively. The inequality to count the number of live digits is as follows:

$$\begin{aligned} &\cdots + 7x_{a,j-C_a-4} + 7x_{a,j-C_a-3} + 5x_{a,j-C_a-2} \\ &+ 3x_{a,j-C_a-1} + x_{a,j-C_a} - x_{a,j-C_a+1} - x_{a,j-C_a+2} \\ -7x_{b,j-5} - 7x_{b,j-4} - 5x_{b,j-3} - 3x_{b,j-2} - x_{b,j-1} + x_{b,j} + x_{b,j+1} \cdots \ &\leq \ 2M_R. \end{aligned} \tag{16}$$

More generally, in the case of overlapped scheduling, the inequality to count the number of live digits at the time class $J = 0, 1, \ldots, T_r - 1$ is

$$\sum_{(a,b)\in E_0} \sum_{k=\underline{k_{ab}}}^{\overline{k_{ab}}} \left\{ \begin{array}{ll} & \displaystyle\sum_{p=0}^{n-\tilde{l}_n T_r - 1} (-2nk + 2p + 1 + 2(p+1)\tilde{l}_n) x_{a,J+kT_r-C_a-p} \\ + & \displaystyle\sum_{p=n-\tilde{l}_n T_r}^{T_r - 1} (-2nk + 2(n - \tilde{l}_n T_r) - 1 + 2(p+1)\tilde{l}_n) x_{a,J+kT_r-C_a-p} \\ + & \displaystyle\sum_{p=1}^{n-l_n T_r} (2n(k+W_e) - 2p + 1 - 2pl_n) x_{b,J+kT_r-p} \\ + & \displaystyle\sum_{p=0}^{T_r-(n-l_nT_r)-1} (2n(k+W_e) + 1 + 2pl_n) x_{b,J+kT_r+p} \end{array} \right\} \leq 2M_R \tag{17}$$



(a)

(b)

Figure 8: Examples of time assignment and life-time of digits ($n = 5$).

where

$$\begin{aligned}
W_e &= \text{the number of delays on the edge } e = (a, b)\\
l_n &= \left\lfloor \frac{n}{T_r} \right\rfloor\\
\tilde{l}_n &= \left\lfloor \frac{n-1}{T_r} \right\rfloor\\
\overline{k_{ab}} &= \max\left\{ \left\lceil \frac{UB_a + C_a + T_r - 1 - J}{T_r} \right\rceil, \left\lceil \frac{UB_b + n - J}{T_r} \right\rceil \right\}\\
\underline{k_{ab}} &= \min\left\{ \left\lceil \frac{LB_a + C_a - J}{T_r} \right\rceil, \left\lceil \frac{LB_b + n - T_r + 1 - J}{T_r} \right\rceil \right\}.
\end{aligned}$$

Example: Fig.8 shows results of the time assignment of the nodes a and b for $T_r = 3$. We assume that $n = 5$, $C_a = 2$, and $W_e = 0$. In the case of the time assignment shown in Fig.8(a), node a and node b are assigned time steps -2 and 0, respectively. Two digit-serial registers must be used at the time class 0 since the life-time of the first and the third digits contain the time class 0. The first term of the left-hand side of (17) becomes 3 when $k = 0$ and $p = 0$ and the fourth term of the left-hand side of (17) becomes 1 when $k = 0$ and $p = 0$. Therefore, in this case, the left-hand side of (17) is 4 which is equal to twice the number of required registers. In Fig.8(b), node a and node b are assigned time steps 0 and 6, respectively. In this case, 7 digit-serial registers must be used at the time class 0 since the time class 0 is contained twice at time step 3, 4 times at time step 6, and once at time step 9. The first term of the left-hand side of (17) becomes -3 when $k = 1$ and $p = 1$ and the third term of the left-hand side of (17) becomes 17 when $k = 2$ and $p = 1$. Therefore, in this case, the left-hand side of (17) is 14 which is also equal to twice the number of required registers.

To simplify notation, assume

$$P_1(k, p, n) \quad = \quad -2nk + 2p + 1 + 2(p+1)\tilde{l}_n, \tag{18}$$

$$P_2(k,p,n) = -2nk + 2(n - \tilde{l}_n T_r) - 1 + 2(p+1)\tilde{l}_n, \tag{19}$$

$$P_3^e(k,p,n) = 2n(k+W_e) - 2p + 1 - 2pl_n, \tag{20}$$

$$P_4^e(k,p,n) = 2n(k+W_e) + 1 + 2pl_n. \tag{21}$$

It is important to note that the following inequality

$$\sum_{(a,b)\in E_0} \sum_{k=\underline{k_{ab}}}^{\overline{k_{ab}}} \left\{ \begin{array}{l} \displaystyle\sum_{p=0}^{n-\tilde{l}_n T_r - 1} (P_1(k,p,n) - S) x_{a,J+kT_r-C_a-p} \\ + \displaystyle\sum_{p=n-\tilde{l}_n T_r}^{T_r-1} (P_2(k,p,n) - S) x_{a,J+kT_r-C_a-p} \\ + \displaystyle\sum_{p=1}^{n-l_n T_r} (P_3^e(k,p,n) + S) x_{b,J+kT_r-p} \\ + \displaystyle\sum_{p=0}^{T_r-(n-l_nT_r)-1} (P_4^e(k,p,n) + S) x_{b,J+kT_r+p} \end{array} \right\} \leq 2M_R, \tag{22}$$

where S is an arbitrary integer, can be used instead of the inequality (17) since $-S$ for variables $x_{a,j}$ and $+S$ for variables $x_{b,j}$ cancel each other. This model is used in section 2.2.3.

2.2.3 Register Minimization in Architectures with Multiple Data Formats

Generally, slower processors are less expensive than faster processors. Therefore, using the slower but less expensive processors for the computations which do not require fast execution may result in a system with lower cost. A processor of one design style inputs data of the format different from the output data of a processor of another design style. If the output data of a bit-parallel processor is input to a bit-serial processor, we must use a data format converter which converts data format from bit-parallel to bit-serial. Such data format converter may be designed as described in [41]. In this section, we show the ILP model to minimize the total cost of processors and converters. Then, the ILP model is extended to minimize the total cost of processors, converters, and registers.

2.2.3.1 ILP Model for Processor Type Selection

We have developed an ILP model for the time assignment supporting the processor type selection. In this ILP model, each node is assigned to a processor type chosen from the library of processor types so that the total cost of processors is minimized without violating any precedence constraints. Data format converters are automatically included in the synthesized architecture when necessary.

Let LB_v^i and UB_v^i denote respectively the lower bound and the upper bound of the time at which a converter of type v could start converting the data output from node i. These are also determined by ASAP and ALAP scheduling results. Let R_v^i denote the scheduling range $[LB_v^i, UB_v^i]$. We define $R_v^i + k$ to denote the closed time interval $[LB_v^i + k, UB_v^i + k]$ for any integers k.

The computation latency and the pipeline period are now specified for each processor type t. If a node is assigned to a processor of type t, its computation latency is C_t and its pipeline period is L_t.

The ILP model minimizes the cost (23), subject to the constraints (24)–(30). The following parameters are used in addition to those defined in section 2.2.1.

$PROC$ is the library of available processors.

F_i denotes the subset of processors $F_i \subset PROC$, capable of executing node $i \in N$.

C_t is the computation latency of a processor of type t.

L_t is the pipeline period of a processor of type t.

m_t is the cost a processor of type t.

G_t is the set of nodes which can be executed on a processor of type t.

$x_{i,j,t}$ is a binary variable. $x_{i,j,t} = 1$ means that node i starts at time step j on a processor of type t.

$FORM$ is the set of input and output formats for all the processors.

$I(t)$ and $O(t)$ are respectively the input and output data formats of processor t.

$CONV$ is the library of available converters.

v_{qr} denotes a data format converter which converts data from format q to format r.

C_v is the conversion latency of a converter of type t.

L_v is the pipeline period of a converter of type t.

m_v is the cost a converter of type t.

V_v is the set of nodes which could be assigned to a processor whose output format is the same as the input data format of a converter of type v.

$y_{i,j,v}$ is a binary variable. $y_{i,j,v} = 1$ means that a data format converter of type v is used and the conversion for the output data of node i starts at time step j.

M_t and M_v are integer variables respectively indicating the number of processors of style t and the number of converters of type v.

$$\text{Minimize } COST = \sum_{t \in PROC} m_t M_t + \sum_{v \in CONV} m_v M_v \tag{23}$$

$$\sum_{t \in F_i} \sum_{j \in R_i} x_{i,j,t} = 1 \qquad \forall i \in N. \tag{24}$$

$$\sum_{j \in R^a_{v_{qr}}} y_{a,j,v_{qr}} \geq \sum_{\substack{ta \in F_a \\ O(ta)=q}} \sum_{ja \in R_a} x_{a,ja,ta} + \sum_{\substack{tb \in F_b \\ I(tb)=r}} \sum_{jb \in R_b} x_{b,jb,tb} - 1 \quad \forall q, r \in FORM, e = (a,b) \in E. \tag{25}$$

$$\sum_{ta \in F_a} \sum_{ja = j - C_{ta} - C_{v_{O(ta),r}} + 1}^{UB_a} x_{a,ja,ta} + \sum_{\substack{tb \in F_b \\ I(tb)=r}} \sum_{jb = LB_b}^{j - W_e T_r} x_{b,jb,tb} \leq 1 \tag{26}$$

$$\forall r \in FORM, e = (a,b) \in E, j \in (R_a + C_{t_a} + C_{v_{O(ta),r1}} - 1) \cap (R_b + W_e T_r).$$

$$\sum_{ta \in F_a} \sum_{ja = j - C_{ta} + 1}^{UB_a} x_{a,ja,ta} + \sum_{\substack{v_{qr_1} \\ r_1 = r}} \sum_{j_1 = LB^a_{v_{qr_1}}}^{j} y_{a,j_1,v_{qr_1}} \leq 1 \tag{27}$$

$$\forall r \in FORM, a \in N, j \in (R_a + \min C_{t_a} - 1) \cap (\cup R^a_{v_{qr}})$$

$$\sum_{\substack{v_{qr_1} \\ r_1 = r}} \sum_{j_1 = j - C_{v_{qr_1}} + 1}^{UB^a_{v_{qr_1}}} y_{a,j_1,v_{qr_1}} + \sum_{\substack{tb \in F_b \\ I(tb)=r}} \sum_{jb = LB_b}^{j - W_e T_r} x_{b,jb,tb} \leq 1 \tag{28}$$

$$\forall r \in FORM, e = (a,b) \in E, j \in (\cup(R^a_{v_{qr}} + C_{v_{qr}} - 1)) \cap (R_b + W_e T_r)$$

$$\sum_{i \in G_t} \left\{ \sum_{k_1 = \left\lfloor \frac{LB_i}{T_r} \right\rfloor}^{\left\lceil \frac{UB_i}{T_r} \right\rceil} \sum_{p=0}^{L_t - \left\lfloor \frac{L_t - 1}{T_r} \right\rfloor T_r - 1} x_{i,J+k_1 T_r - p,t} + \left\lfloor \frac{L_t - 1}{T_r} \right\rfloor \sum_{j_1 \in R_i} x_{i,j_1,t} \right\} \leq M_t \tag{29}$$

$$\forall J = 0, 1, \ldots, T_r - 1,\ t \in PROC$$

$$\sum_{i \in V_v} \left\{ \sum_{k_1 = \left\lfloor \frac{LB^i_v}{T_r} \right\rfloor}^{\left\lceil \frac{UB^i_v}{T_r} \right\rceil} \sum_{p=0}^{L_v - \left\lfloor \frac{L_v - 1}{T_r} \right\rfloor T_r - 1} y_{i,J+k_1 T_r - p,v} + \left\lfloor \frac{L_v - 1}{T_r} \right\rfloor \sum_{j_1 \in R^i_v} y_{i,j_1,v} \right\} \leq M_v \tag{30}$$

$$\forall J = 0, 1, \ldots, T_r - 1,\ v \in CONV.$$

The node assignment constraint (24) ensures that node i has one start time and is assigned to one processor. The converter assignment constraint (25) ensures that a data format converter of type v_{qr} is used if node a is assigned to a processor whose output data format is

q and whose immediate successor node b is assigned to a processor whose input data format is r.

In the precedence constraint from processor to processor (26), the data format conversion time is taken into account. If an edge $e = (a, b)$ exists, the computation of node b must start at least $C_{t_a} + C_{v_{O(ta),I(tb)}} - W_e T_r$ time step later than the computation of node a starts since the computation of node a takes C_{t_a} time steps and the data format conversion takes $C_{v_{O(ta),I(tb)}}$ time steps. If $O(t_a) = I(t_b)$, no data format conversion is performed since $C_{v_{rr}} = 0$ for $r \in FORM$.

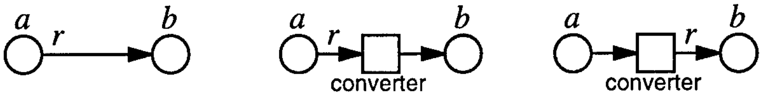
Inequalities (27) and (28) ensure the precedence constraints from processor to converter and from converter to processor, respectively. In the case when the output format of the converter and the input format of the processor are different, there is no need to constrain the precedence relation between them. In that case, at least one of the terms on the left-hand side of the inequality (28) is 0 and the inequality is automatically satisfied.

Inequalities (29) and (30) are used to count the number of processors and the number of converters of each type.

2.2.3.2 Counting the Number of Registers During the Time Assignment

To calculate the cost of registers, we must know the exact number of registers, M_{R_r}, of each data format r. Although it is possible to use part of a bit-parallel register as a bit-serial register and vice versa, it would require complex control circuits. Therefore, we assume that no 1-bit register is commonly used as part of registers of different data formats. That is, bit-parallel registers are always used as bit-parallel registers and bit-serial registers are always used as bit-serial registers. Henceforth, we will count the number of registers of each data format separately and just sum them up with cost factors to calculate the total cost of registers.

Registers of the data format r are used for the edge (a, b) in the cases where (i) node a and node b are assigned to processors of data format r (as illustrated in Fig.9(a)), (ii) node a is assigned to a processor of data format r, node b is assigned to a processor of data format f other than r, and the data format converter of type v_{rf} is used to convert the data output from node a (Fig.9(b)), or (iii) node a is assigned to a processor of data format q other than



(a) (b) (c)

Figure 9: Assignment of nodes to processor types.

r, node b is assigned to a processor of data format r, and the data format converter of type v_{qr} is used to convert the data output from node a (Fig.9(c)). There also exists a case where no register of data format r is used for the edge (a, b). In that case, both the nodes a and b are assigned to processors of data format other than r. Which case really occurs depends on the processor type selection and cannot be known prior to solving the ILP model.

In the case where node b is the only immediate successor node of node a, at most one converter of type v_{qr} (for any q) is used if the node b is assigned to a processor of data format r. The required number of format r registers for the output data of node a in the case where node a has only one immediate successor node, $M_R(a, r, J)$, is calculated as

$$2M_R(a,r,J) = \sum_{k=\underline{k_{ab}}}^{\overline{k_{ab}}} \left\{ \begin{array}{l} \displaystyle\sum_{p=0}^{n_r-\tilde{l}_{n_r}T_r-1} (P_1(k,p,n_r) - S^r_{ab})\Big(\sum_{\substack{t_a\in F_a\\ O(t_a)=r}} x_{a,J+kT_r-C_{t_a}-p,t_a} + \sum_{\substack{v_{qr_1}\\ r_1=r}} y_{a,J+kT_r-C_{v_{qr_1}}-p,v_{qr_1}}\Big) \\ \displaystyle + \sum_{p=n_r-\tilde{l}_{n_r}T_r}^{T_r-1} (P_2(k,p,n_r) - S^r_{ab})\Big(\sum_{\substack{t_a\in F_a\\ O(t_a)=r}} x_{a,J+kT_r-C_{t_a}-p,t_a} + \sum_{\substack{v_{qr_1}\\ r_1=r}} y_{a,J+kT_r-C_{v_{qr_1}}-p,v_{qr_1}}\Big) \\ \displaystyle + \sum_{p=1}^{n_r-l_{n_r}T_r} (P^e_3(k,p,n_r) + S^r_{ab})\Big(\sum_{\substack{t_b\in F_b\\ I(t_b)=r}} x_{b,J+kT_r-p,t_b} + \sum_{\substack{v_{r_1f}\\ r_1=r}} y_{a,J+(k+W_e)T_r-p,v_{r_1f}}\Big) \\ \displaystyle + \sum_{p=0}^{T_r-(n_r-l_{n_r}T_r)-1} (P^e_4(k,p,n_r) + S^r_{ab})\Big(\sum_{\substack{t_b\in F_b\\ I(t_b)=r}} x_{b,J+kT_r+p,t_b} + \sum_{\substack{v_{r_1f}\\ r_1=r}} y_{a,J+(k+W_e)T_r+p,v_{r_1f}}\Big) \end{array} \right\} \tag{31}$$

where n_r is the number of digits of data format r. The integer S^r_{ab} is chosen so that every coefficient, $P^e_3(k,p,n_r) + S^r_{ab}$ or $P^e_4(k,p,n_r) + S^r_{ab}$, in the third and the fourth terms of the right-hand side is positive. In that case, the coefficients $P_1(k,p,n_r) - S^r_{ab}$ and $P_2(k,p,n_r) - S^r_{ab}$ in the first and the second terms of the right-hand side may be positive, negative, or zero. Hence, binary variables $y_{a,j,v_{qr}}$ may unnecessarily become 1 to falsely reduce the value of the right-hand side of (31) and give an incorrect number of registers. To prevent this, the constraints

$$\sum_{\substack{v_{qr_1} \\ r_1=r}} \sum_{j \in R^a_{v_{qr_1}}} y_{a,j,v_{qr_1}} + \sum_{\substack{t_a \in F_a \\ O(t_a)=r}} \sum_{ja \in R_a} x_{a,ja,t_a} \leq 1 \quad \forall a \in N, \tag{32}$$

$$\sum_{\substack{v_{qr_1} \\ r_1=r}} \sum_{j \in R^a_{v_{r_1 f}}} y_{a,j,v_{r_1 f}} \leq \sum_{\substack{t_b \in F_b \\ I(t_b)=r}} \sum_{jb \in R_b} x_{b,jb,t_b} \quad \forall (a,b) \in E \tag{33}$$

are introduced so that variables $y_{a,j,v_{qr}}$ do not unnecessarily become 1. Note that the constraints (32) and (33) do not eliminate any assignment possibility. If node a is assigned to a processor of format r (the second term on the right-hand side of (32) is 1), then we need not use a data format converter which converts the output data of node a into format r. Moreover, if node b is assigned to a processor which does not input data of format r (the right-hand side of (33) is 0), then we also need not use a data format converter which converts data into format r. Therefore, these constraint may be satisfied in the case where the converters are inserted properly.

On the other hand, in the case where node a has more than 1 immediate successor nodes, we use another binary variables $g_{a,j,r}$. It is important to note that the transitivity analysis is of no use in the case of multiple data formats. It is because which immediate successor node last uses the format r version of data can not be known until nodes are assigned to processor types. Therefore, we introduce new variables $g_{a,j,r}$ rather than using very large number of constraints for counting the number of registers. Although the number of variables would be increased, the number of constraints is greatly decreased from $\prod_{a \in N} s_a * T_r$ to T_r for every data format $r \in FORM$.

The variable $g_{a,j,r} = 1$ means that the format r version of the output data of node a is last used at time step j. If such format r version of data is not used, all the variables are 0. To compute the value of $g_{a,j,r}$, we use the following inequalities

$$\sum_{j \in R^a_r} j \cdot g_{a,j,r} \geq \sum_{\substack{t_b \in F_b \\ I(t_b)=r}} \sum_{j \in R_b} (j + W_e) \cdot x_{b,j,t_b}, \quad \forall a \in N_m \, e = (a,b) \in E \tag{34}$$

$$\sum_{j \in R^a_r} j \cdot g_{a,j,r} \geq \sum_{\substack{v_{qr_1} \\ r_1=r}} \sum_{j \in R^a_{v_{qr_1}}} j \cdot y_{a,j,v_{qr_1}}, \quad \forall a \in N_m \tag{35}$$

$$\sum_{j \in R^a_r} g_{a,j,r} \leq 1, \quad \forall a \in N_m \tag{36}$$

where N_m is the set of node with more than one immediate successor nodes and R_r^a is the union of the scheduling ranges R_a and $R_{v_{qr}}^a$.

Then the required number of format r registers for the output data of node a in the case where node a has more than one immediate successor nodes is calculated as

$$2M_R'(a,r,J) = \sum_{k=\underline{k_{ab}}}^{\overline{k_{ab}}} \left\{ \begin{array}{l} \displaystyle\sum_{p=0}^{n_r-\tilde{l}_{n_r}T_r-1} (P_1(k,p,n_r) - S_{ab}^r)\Big(\sum_{\substack{t_a\in F_a \\ O(t_a)=r}} x_{a,J+kT_r-C_{t_a}-p,t_a} + \sum_{\substack{v_{qr_1} \\ r_1=r}} y_{a,J+kT_r-C_{v_{qr_1}}-p,v_{qr_1}} \Big) \\ \displaystyle + \sum_{p=n_r-\tilde{l}_{n_r}T_r}^{T_r-1} (P_2(k,p,n_r) - S_{ab}^r)\Big(\sum_{\substack{t_a\in F_a \\ O(t_a)=r}} x_{a,J+kT_r-C_{t_a}-p,t_a} + \sum_{\substack{v_{qr_1} \\ r_1=r}} y_{a,J+kT_r-C_{v_{qr_1}}-p,v_{qr_1}} \Big) \\ \displaystyle + \sum_{p=1}^{n_r-l_{n_r}T_r} (P_3^e(k,p,n_r) + S_{ab}^r) g_{a,J+kT_r-p,r} \\ \displaystyle + \sum_{p=0}^{T_r-(n_r-l_{n_r}T_r)-1} (P_4^e(k,p,n_r) + S_{ab}^r) g_{a,J+kT_r+p,r} \end{array} \right\}. \tag{37}$$

The integer S_{ab}^r is chosen so that every coefficient, $P_3^e(k,p,n_r) + S_{ab}^r$ or $P_4^e(k,p,n_r) + S_{ab}^r$, in the third and the fourth terms of the right-hand side is positive. Thus, the coefficients $P_1(k,p,n_r) - S_{ab}^r$ and $P_2(k,p,n_r) - S_{ab}^r$ in the first and the second terms of the right-hand side may be negative. We must use the constraints (32) and (33) so that converters are not unnecessarily used.

The ILP model to synthesize the architecture with lowest cost of processors, converters, and registers minimizes the cost (38), subject to the constraints (24)–(30), (32), (33), (34)–(36), and (39). Here, M_r is the number of format r registers and m_r is the relative cost of a format r register.

$$\text{Minimize } COST = \sum_{t\in PROC} m_t M_t + \sum_{v\in CONV} m_v M_v + \sum_{r\in FORM} m_r M_r \tag{38}$$

$$\sum_{a\in N-N_m} 2M_R(a,r,J) + \sum_{a\in N_m} 2M_R'(a,r,J) \le 2M_r \quad \forall r \in FORM,\ J = 0,1,\ldots,T_r-1. \tag{39}$$

2.2.4 Experimental Result

In this experiment, the effectiveness of the ILP model to minimize the cost of registers as well as the cost of processors and converters is confirmed. We use a DFG of a biquad filer illustrated in Fig.10.

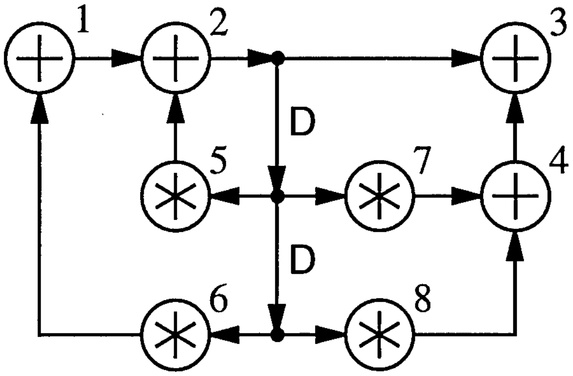


Figure 10: A data flow graph of a biquad filter.

Table 7 shows a library of processor types. Each node in the DFG can be assigned to one or more of the processors in the library. In Table 7, the computational latency, C, the pipeline period, L, the input and output data format, I and O, and the cost, m, are shown for each processor type. Processors A1 and A2 represent two different adder implementations while processors M3 and M4 represent two different multiplier implementations. Processors A1 and M3 input and output data of format *bp* and Processors A2 and M4 input and output data of format *hp*. These formats *bp* and *hp* imply the bit-parallel and the half-word parallel, respectively. The half-word parallel data format is the digit-serial where the digit-size is half the wordlength. Therefore, half the bits of one word are processed at the same time and the number of digits of one word of half-word parallel is two. For the DFG of the biquad filter shown in Fig.10, Nodes 1, 2, 3, and 4 can be assigned to either processor A1 or processor A2 in Table 7. Similarly, nodes 5, 6, 7, and 8 can be assigned to either processor M3 or M4 in Table 7.

Furthermore to support data format conversion, we include a library of data format converters which convert between all possible data formats listed in the library of processors. For example the library of processors in Table 7 requires two data format converters as shown in Table 8. Each of the data format converters is classified according to its conversion type, its conversion latency, C, its pipeline period, L, and its cost, m. The conversion latency of the converter of type $bp \rightarrow hp$ is 0 since the first digit of the converted data is available at the time when the bit-parallel data is input to the converter.

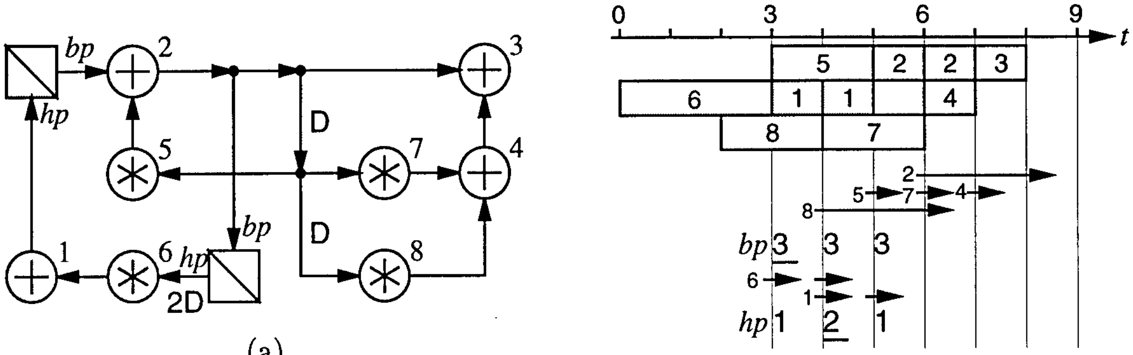
We choose the costs of a register as $m_{bp} = 2$ and $m_{hp} = 1$.

Table 7: Processor specifications

type	C	L	I	O	m
A1	1	1	*bp*	*bp*	10
A2	1	2	*hp*	*hp*	5
M3	2	2	*bp*	*bp*	50
M4	3	3	*hp*	*hp*	25

Table 8: Data format converter specifications

type	conversion	C	L	m
$v_{hp,bp}$	$bp \rightarrow hp$	0	1	1
$v_{bp,hp}$	$hp \rightarrow bp$	1	1	1

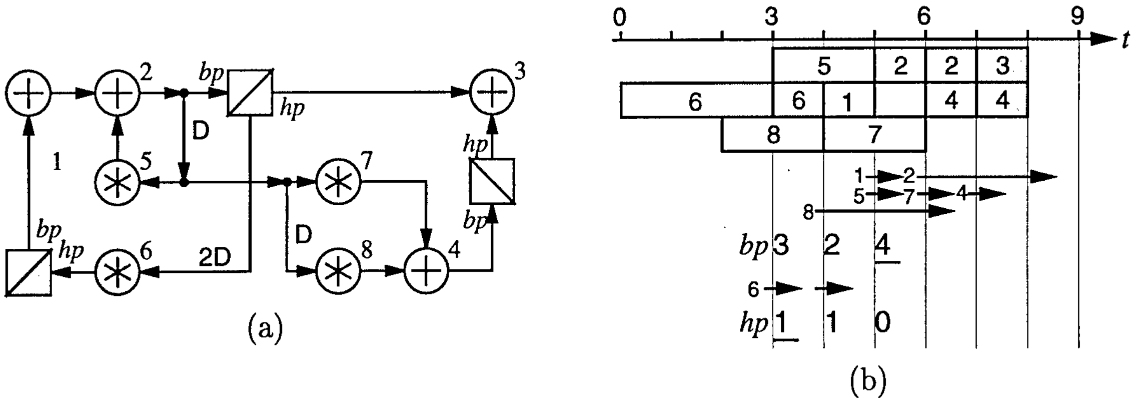


(a)

(b)

Figure 11: Time assignment result by the complete ILP model. (a) The assignment of nodes to processor types. (b) Time chart of the time assignment and life-time of data.

Fig.11 shows a time assignment result for the iteration period $T_r = 3$ obtained by solving the complete ILP model with register minimization. Fig.11(a) shows the assignment between nodes and processors and inserted converters. A white node means it is assigned to either an A1 adder or an M3 multiplier. A dotted node means it is assigned to either an A2 adder or an M4 multiplier. Boxes are then inserted to represent data format converters. The time chart of the node computations and data format conversions and the life-time of data are illustrated in Fig.11(b). In this figure, a box represents either a computation of node or a data format conversion. An arrow represents the life-time of a data in the case of format *bp* or a digit in the case of format *hp*. For example, the computation of node 5 starts at time step 3 and its result is output at time step 5 since the computation latency of M3 multiplier is 2. That result is stored in a register of format *bp* at the time step 5 and used by the computation of node 2 at time step 5. A data format conversion of the type $bp \rightarrow hp$ for the output data of node 2 (represented by a half shaded box with '2' inside) is executed at time step 6. The first digit of the converted data is output immediately at time step 6 and used by node 6. The second half of the data is stored in the converter and output as the



(a)

(b)

Figure 12: Time assignment result with ILP model division. (a) The assignment of nodes to processor types. (b) Time chart of the time assignment and life-time of data.

second digit at at time step 7. Then it is input by node 6. In this case, 1 A1 adder, 1 A2 adder, 2 M3 multiplier2, 1 M4 multipliers, 1 *bp* → *hp* converter, and 1 *hp* → *bp* converter, 3 *bp* registers, and 2 *hp* registers are used in this architecture of the lowest cost of 150.

Fig. 12 shows a time assignment result for the iteration period $T_r = 3$ obtained by solving the divided ILP models. In this case, the cost of processors and converters is the same as the result by the complete model. However, we need 4 *bp* registers and 1 *hp* register and the total cost is 151. This cost is one unit of cost higher than the optimal result obtained by the complete ILP model. This is because the assignment of nodes to processor types is fixed as obtained by the second ILP model and there is no chance to alter the assignment while the cost of regisers is precisely calculated and minimized by the third ILP model.

Table 9 compares the complete ILP model and the divided ILP models. Table 9 shows the number of constraints (eqn) and the number of variables (var) in the ILP model, the cost of synthesized architecture, and the CPU time to solve the ILP model. The CPU times are measured by the ILP solver GAMS/OSL [42] running on a SparcStation 20. While the CPU time to solve the complete ILP model is 134 seconds, the total CPU time for the divided ILP models is only 5.5 seconds. Thus, the divided ILP models save much CPU time at the expense of 0.7% increase in the cost of synthesized architecture.

For more practical results, we have synthesized architectures for some benchmark dataflow graphs. In this case, we assume the library of processors and the library of converters as shown in Tables 10 and 11. We also assume that arithmetic is in fixed point and the

Table 9: The ILP Models for the Biquad Filter Synthesis

Model	eqn	var	Cost	CPU [sec]
complete	374	231	150	134.06
first	65	72	142	2.80
second	81	84	150	2.23
third	81	69	151	0.47

Table 10: Library of Processor Types (wordlength = 16)

type	processor	C	L	m	I	O
A_{bp}	Bit-parallel adder	1	1	53	bp	bp
A_{hp}	Half-word parallel adder	1	2	19	hp	hp
A_{ds}	4-bit digit-serial adder	1	4	6	ds	ds
M_{bp}	Bit-parallel multiplier	5	1	331	bp	bp
M_{hp}	Half-word parallel multiplier	6	2	173	hp	hp
M_{ds}	4-bit digit-serial multiplier	9	5	86	ds	ds

wordlength is 16 bits. The format *ds* implies the 4-bit digit-serial where the digit size is 4 bits. Table 12 shows the specification of the register of each data format. In this table n is the number of digits of one word and m is the cost of one register of each data format.

Table 13 shows; the data-flow graph; the specified iteration period T_r; the model; the number of constraints and the number of varibles of the ILP model; CPU time in seconds to solve the ILP model; the lowest cost architecture; the number of registers; and the total cost. The ILP models are solve by the ILP solver GAMS/OSL running on a SparcStation 2. For example, in the case of the 4th order lattice filter with $T_r = 14$, the second ILP model is not used. This is because only one type of processor is used for each operation type (addition or multiplication) and therefore the assignment of node computations to processor types is

Table 11: Converter Types

type	conversion	C	L	m
$v_{bp,hp}$	$bp \rightarrow hp$	0	1	3
$v_{bp,ds}$	$bp \rightarrow ds$	0	3	4
$v_{hp,bp}$	$hp \rightarrow bp$	1	1	3
$v_{hp,ds}$	$hp \rightarrow ds$	0	2	3
$v_{ds,bp}$	$ds \rightarrow bp$	3	3	4
$v_{ds,hp}$	$ds \rightarrow hp$	2	2	3

Table 12: Registers

fmt	n	m
bp	1	8
hp	2	4
ds	4	2

Table 13: Time Assignment Benchmarks

DFG	T_r	Mdl	eqn	var	CPU	Lowest cost architecture	reg	Cost
4th Order Lattice Filter	14	1st	49	47	0.82	$3A_{bp}$, $2M_{bp}$		821
		3rd	41	28	0.86	$2A_{bp}$, M_{bp}	$5R_{bp}$	861
	15	1st	138	84	2.14	$2A_{bp}$, M_{bp}		437
		3rd	119	114	2.26	$2A_{bp}$, M_{bp}	$5R_{bp}$	477
	16	1st	212	129	14.1	A_{bp}, A_{hp}, M_{bp}, $v_{bp,hp}$, $v_{hp,bp}$		409
		2nd	207	130	17.7	A_{bp}, A_{hp}, M_{bp}, $v_{bp,hp}$, $v_{hp,bp}$	$5R_{bp}$	449
		3rd	235	146	5.74	A_{bp}, A_{hp}, M_{bp}, $2v_{bp,hp}$, $v_{hp,bp}$	$3R_{bp}$, $3R_{hp}$	448
	17	1st	310	181	17.6	A_{bp}, M_{bp}		437
		3rd	163	114	17.0	A_{bp}, M_{bp}	$6R_{bp}$	432
	18	1st	276	154	5.96	A_{hp}, A_{ds}, M_{hp}, $v_{hp,ds}$, $v_{ds,hp}$		223
		2nd	206	122	4.37	A_{hp}, A_{ds}, M_{hp}, $v_{hp,ds}$, $v_{ds,hp}$	$9R_{hp}$	259
		3rd	225	130	1.24	A_{hp}, A_{ds}, M_{hp}, $v_{hp,ds}$, $v_{ds,hp}$	$7R_{hp}$, $4R_{ds}$	259
5th Order Elliptic Wave Filter	25	1st	243	167	2.02	$3A_{bp}$, M_{bp}		490
		3rd	141	113	1.39	$3A_{bp}$, M_{bp}	$9R_{bp}$	562
	26	1st	415	249	24.4	$2A_{bp}$, M_{bp}		437
		3rd	185	142	2.47	$2A_{bp}$, M_{bp}	$9R_{bp}$	509
	27	1st	586	326	651	A_{bp}, A_{hp}, M_{bp}, $v_{bp,hp}$, $v_{hp,bp}$		428
		2nd	452	288	806	A_{bp}, A_{hp}, M_{bp}, $v_{bp,hp}$, $v_{hp,bp}$	$9R_{bp}$	500
		3rd	494	367	13.0	A_{bp}, A_{hp}, M_{bp}, $2v_{bp,hp}$, $v_{hp,bp}$	$6R_{bp}$, $5R_{hp}$	499
4th Order Jaumann Filter	16	1st	346	291	9.53	$2A_{bp}$, M_{bp}		437
		3rd	232	154	4.32	$2A_{bp}$, M_{bp}	$6R_{bp}$	485
	17	1st	417	327	13.5	A_{bp}, M_{bp}		384
		3rd	254	172	7.45	A_{bp}, M_{bp}	$6R_{bp}$	432
	18	1st	451	362	24.3	A_{bp}, M_{bp}		384
		3rd	276	190	12.9	A_{bp}, M_{bp}	$6R_{bp}$	432
	17	1st	305	262	7.12	$2A_{hp}$, M_{hp}		211
		3rd	291	191	5.92	$2A_{hp}$, M_{hp}	$13R_{hp}$	263
	18	1st	348	291	17.0	$2A_{hp}$, M_{hp}		211
		3rd	314	209	200	$2A_{hp}$, M_{hp}	$13R_{hp}$	263
4-stage Pipelined Lattice Filter	3	1st	146	145	5.55	$2A_{bp}$, $7A_{ds}$, $5M_{bp}$, $v_{bp,ds}$		1807
		2nd	254	133	1710	$2A_{bp}$, $8A_{ds}$, $5M_{bp}$, $v_{bp,ds}$	$13R_{bp}$	1917
		3rd	229	135	5.74	$2A_{bp}$, $8A_{ds}$, $5M_{bp}$, $7v_{bp,ds}$	$11R_{bp}$, $8R_{ds}$	1941
	4	1st	215	210	4.42	A_{hp}, $9A_{ds}$, $4M_{bp}$, $v_{bp,hp}$, $v_{bp,ds}$, $v_{hp,bp}$, $v_{ds,bp}$		1411
		2nd	373	178	4359	A_{hp}, $9A_{ds}$, $4M_{bp}$, $v_{bp,hp}$, $v_{bp,ds}$, $v_{hp,bp}$, $v_{ds,bp}$	$12R_{bp}$	1513
		3rd	437	230	55.7	A_{hp}, $9A_{ds}$, $4M_{bp}$, $2v_{bp,hp}$, $6v_{bp,ds}$, $v_{hp,bp}$, $v_{ds,bp}$	$8R_{bp}$, $2R_{hp}$, $8R_{ds}$	1528
	5	1st	238	261	14.8	$9A_{ds}$, $3M_{bp}$, $v_{bp,ds}$, $v_{ds,bp}$		1055
		3rd	534	276	312	$9A_{bp}$, $3M_{bp}$, $9v_{bp,ds}$, $v_{ds,bp}$	$6R_{bp}$, $24R_{ds}$	1187
16 Point Fir Filter	1	1st	96	64	1.09	$60A_{ds}$, $8M_{bp}$, $v_{bp,ds}$, $v_{ds,bp}$		3016
		3rd	134	84	0.72	$60A_{ds}$, $8M_{bp}$, $24v_{bp,ds}$, $24v_{ds,bp}$	$8R_{bp}$, $56R_{ds}$	3376
	2	1st	100	117	1.12	$30A_{ds}$, $4M_{bp}$, $v_{bp,ds}$, $v_{ds,bp}$		1508
		3rd	188	120	2.71	$30A_{ds}$, $4M_{bp}$, $12v_{bp,ds}$, $12v_{ds,bp}$	$4R_{bp}$, $30R_{ds}$	1692
	3	1st	104	170	1.82	$20A_{ds}$, $3M_{bp}$, $v_{bp,ds}$, $v_{ds,bp}$		1121
		3rd	248	160	38.0	$20A_{ds}$, $3M_{bp}$, $8v_{bp,ds}$, $8v_{ds,bp}$	$3R_{bp}$, $22R_{ds}$	1245

obvious. Thus, we immediately generate the third ILP model based on the result of the first ILP model. The same applies to other cases where the second ILP model is missing.

3 Other High-Level Tools

We have also developed other tools and methodologies during our pursuit of solutions to the high-level synthesis problem and in developing efficient architectures. In this section we present these new results.

3.1 Determination of Minimum Iteration Period

DSP algorithms are repetitive in nature and can be easily described by iterative data-flow graphs (DFGs) where nodes represent tasks and edges represent communication [43, 44]. Execution of all nodes of the DFG once completes an *iteration.* Successive iterations of any node are executed with a time displacement referred to as the *iteration period.* For all recursive signal processing algorithms, there exists an inherent fundamental lower bound on the iteration period referred to as the *iteration period bound* or simply the iteration bound [45, 46, 47]. This bound is fundamental to an algorithm and is independent of the implementation architecture. In other words, it is impossible to achieve an iteration period less than the bound even when infinite processors are available to execute the recursive algorithm.

Determination of the iteration bound of the data-flow graph is an important problem. First it discourages the designer to attempt to design an architecture with an iteration period less than the iteration bound. Second, the iteration bound needs to be determined in rate-optimal scheduling of iterative data-flow graphs. A schedule is said to be rate-optimal if the iteration period is same as the iteration bound, i.e., the schedule achieves the highest possible rate of operation of the algorithm.

Two algorithms have been recently proposed to determine the iteration bound. A method based on the negative cycle detection was reported in [48] to determine the iteration bound with polynomial time complexity with respect to the number of nodes in the processing algorithm. Another method based on the first-order longest path matrix was proposed

```
Input:   DFG $G = (N, E, q, d)$.
Output:  The iteration bound $T_i$.
1.  Construct the graph $\bar{G}_d = (D, E_d, \bar{w})$ from the given DFG $G = (N, E, q, d)$.
2.  Run the minimum cycle mean algorithm on $\bar{G}_d$.
      Minimum cycle mean algorithm
      2.0  Choose one node $s \in D$ arbitrarily.
      2.1  Calculate the minimum weight $F_k(v)$ of an edge progression
           of length $k$ from $s$ to $v$ as
             $F_k(v) = \min_{(u,v) \in E_d} \{F_{k-1}(u) + \bar{w}(u,v)\}$    for $k = 1, 2, \ldots, |D|$
           with the initial conditions $F_0(s) = 0$; $F_0(v) = \infty$, $v \neq s$.
      2.2  Calculate the minimum cycle mean $\lambda$ of $\bar{G}_d$.
             $\lambda = \min_{v \in D} \max_{0 \le k \le |D|-1} \frac{F_{|D|}(v) - F_k(v)}{|D| - k}$
3.  Now, $T_i = -\lambda$ is the iteration bound of the DFG $G$.
```

Figure 13: The algorithm to determine the iteration bound.

in [49] to determine the lower bound with polynomial time complexity with respect to the number of delays in the processing algorithm. In this section, we propose yet another method based on the *minimum cycle mean algorithm* to determine the iteration bound with lower polynomial time complexity than in [48] and [49].

3.1.1 A New Algorithm to Determine the Iteration Bound

In this section, we describe an algorithm that determines the iteration bound by using the minimum cycle mean algorithm. The *cycle mean* of a cycle c, $m(c)$, is defined as

$$m(c) = \frac{\sum_{e \in c} w(e)}{p_c} \tag{40}$$

where $w(e)$ is the weight of the edge e and p_c is the number of edges in cycle c. In other words, the cycle mean of a cycle c is average weight of the edges included in c.

The *minimum cycle mean problem* involves the determination of the minimum cycle mean, λ, of all the cycles in the given digraph where

$$\lambda = \min_{c} m(c). \tag{41}$$

An efficient algorithm was proposed in [50] to determine the minimum cycle mean for a given graph with time complexity $\mathcal{O}(|N||E|)$, where N and E are the set of nodes and the set of edges of the graph, respectively.

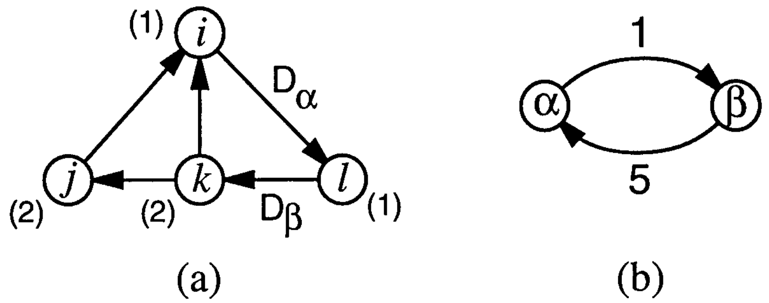


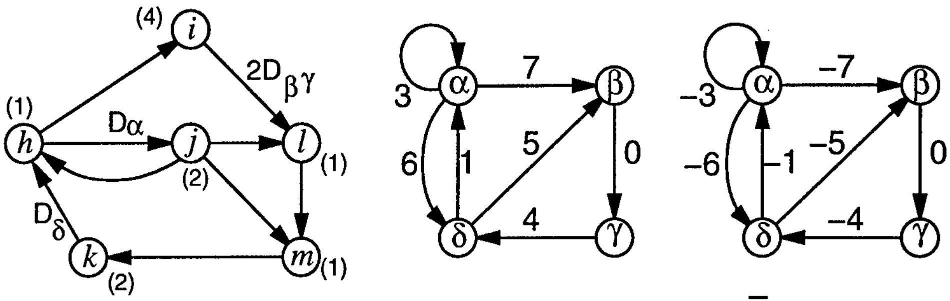
Figure 14: The cycle mean and the cycle bound.

The number of nodes in a cycle is equal to the number of edges of the cycle. According to the definition of the graph $G_d = (D, E_d, w)$, each node in G_d corresponds to a delay in the DFG, G, and the edge weight $w(d_1, d_2)$ of the edge $(d_1, d_2) \in E_d$ is the largest weight among all the paths from the delay d_1 to the delay d_2. Therefore, the cycle mean of the cycle c_d, containing k nodes, d_1, d_2, …, d_k, is the maximum cycle bound of the cycles of G, which contain the delays labeled d_1, d_2, …, d_k. For example, in the graph shown in Fig.14(a), there are two delays labeled α and β, respectively. There exist two cycles $\{(l,k),(k,i),(i,l)\}$ and $\{(l,k),(k,j),(j,i),(i,l)\}$, both of which go through delays α and β. Their cycle bounds are $4/2 = 2$ and $6/2 = 3$, respectively, and the maximum of them is 3. Fig.14(b) shows the graph $G_d = (D, E_d, w)$ corresponding to the graph shown in Fig.14(a). In Fig.14(b), $D = \{\alpha, \beta\}$, $w(\alpha, \beta) = 1$, and $w(\beta, \alpha) = 5$. There exists one cycle $\{(\alpha,\beta),(\beta,\alpha)\}$ and its cycle mean is 3. It equals the maximum cycle bound of the cycles in the graph shown in Fig.14(a), which contain the delays α and β.

Since the cycle mean of a cycle c in the graph G_d equals the maximum cycle bound of the cycles in G which contain the delays in cycle c, the maximum cycle mean of the graph G_d equals the maximum cycle bound of all the cycles in the graph G. Therefore, the iteration bound of the graph G can be obtained as the maximum cycle mean of the graph G_d.

Let C_d denote the set of cycles in graph G_d. Then, the maximum cycle mean of the graph G_d is

$$\begin{aligned}\max_{c \in C_d} m(c) &= \max_{c \in C_d} \frac{\sum_{e \in c} w(e)}{p_c} \\ &= \max_{c \in C_d} \frac{-\sum_{e \in c}(-w(e))}{p_c}\end{aligned}$$



G=(N,E,q,d)
N={h,i,j,k,l,m}
(a)

Gd=(D,Ed,w)
D=(α,β,γ,δ)
(b)

$\bar{G}d$=(D,Ed,$\bar{w}$)
D=(α,β,γ,δ)
(c)

Figure 15: The DFG G and the corresponding edge-weighted digraph G_d. In parenthesis in G are the computation times of nodes.

$$= -\min_{c\in C_d} \frac{\sum_{e\in c}(-w(e))}{p_c}. \tag{42}$$

It is the negative of the minimum cycle mean of the graph $\bar{G}_d = (D, E_d, \bar{w})$, where $\bar{w}(e) = -w(e)$ for every edge $e \in E_d$. Consequently, the maximum cycle mean of the graph G_d, i.e., the iteration bound of the graph G, can be obtained as the negative of the minimum cycle mean of the graph $\bar{G}_d$.

The algorithm to determine the iteration bound of the given graph by means of the minimum cycle mean is summarized in Fig.13.

From the DFG $G = (N, E, q, d)$, constructing $G_d = (D, E_d, w)$ and $\bar{G}_d = (D, E_d, \bar{w})$ requires the computation time of $\mathcal{O}(|D||E|)$ complexity. The time complexity to calculate the minimum cycle mean for the graph $\bar{G}_d = (D, E_d, \bar{w})$ is $\mathcal{O}(|D||E_d|)$. Hence, the total time complexity to determine the iteration bound is $\mathcal{O}(|D||E_d| + |D||E|)$. This time complexity is better than the $\mathcal{O}(|D|^3 \log |D| + |D||E|)$ complexity of the other methods since $|E_d| \leq |D|^2$ and therefore $|E_d| \leq |D|^2 \log |D|$ always hold. The memory requirement for calculating the edge weight w and determining the minimum cycle mean for the graph G_d are $\mathcal{O}(|N|)$ and $\mathcal{O}(|D|^2)$, respectively. The total memory requirement is $\mathcal{O}(|N| + |D|^2)$.

Example. From the given DFG G illustrated in Fig.15(a), the edge-weighted digraph G_d and $\bar{G}_d$ are constructed as shown in Fig.15(b) and (c), respectively. If we choose α as s

Table 14: Comparison of Iteration Bound Determination Algorithms

Method	Time complexity	Memory requirement	CPU [mS]	
			EWF	PLF
NCD	$\mathcal{O}(\lvert N\rvert\lvert E\rvert\log\lvert N\rvert)$	$\mathcal{O}(\lvert N\rvert+\lvert E\rvert)$	25.2[a]	1.00[c]
LPM	$\mathcal{O}(\lvert D\rvert\lvert E\rvert+\lvert D\rvert^4)$	$\mathcal{O}(\lvert N\rvert+\lvert D\rvert^2)$	1.92[b]	2.97[d]
LPM'	$\mathcal{O}(\lvert D\rvert\lvert E\rvert+\lvert D\rvert^3\log\lvert D\rvert)$	$\mathcal{O}(\lvert N\rvert+\lvert D\rvert^2)$	3.58[a]	6.38[c]
MCM	$\mathcal{O}(\lvert D\rvert\lvert E\rvert+\lvert D\rvert\lvert E_d\rvert)$	$\mathcal{O}(\lvert N\rvert+\lvert D\rvert^2)$	0.717[b]	0.650[d]

[a] the obtained iteration bound = 16.0002594 [c] the obtained iteration bound = 1.50439453
[b] the obtained iteration bound = 16.0000000 [d] the obtained iteration bound = 1.50000000

used in the minimum cycle mean algorithm, $F_k(v)$, the minimum weight of paths consisting of exactly k edges in E_d, and $\max_{0\le k\le|D|-1}\frac{F_{|D|}(v)-F_k(v)}{|D|-k}$ are calculated as follows: Then,

$$\begin{array}{cc|ccccc|c} & & & & k & & & \\ & F_k(v) & 0 & 1 & 2 & 3 & 4 & \max\limits_{0\le k\le 3}\dfrac{F_4(v)-F_k(v)}{4-k} \\ \hline & \alpha & 0 & -3 & -7 & -10 & -14 & -3.5 \\ v & \beta & \infty & -7 & -11 & -14 & -18 & -3.5 \\ & \gamma & \infty & \infty & -7 & -11 & -14 & -3 \\ & \delta & \infty & -6 & -9 & -13 & -16 & -3 \end{array} \tag{43}$$

$\lambda=\min_{v\in D}\max_{0\le k\le|D|-1}\frac{F_{|D|}(v)-F_k(v)}{|D|-k}=-3.5$ and the iteration bound of the DFG G is 3.5. The reader may confirm that the critical cycle is $\{(h,j),(j,l),(l,m),(m,k),(k,h)\}$ and its cycle bound, that is the iteration bound of the DFG, is 3.5 since the sum of computation times of nodes h,j,l,m,k is 7, the critical cycle contains 2 delays labeled as α and δ, and $7/2=3.5$.

3.1.2 Experimental Results

The CPU time to determine the iteration bound for practical DFGs are compared. We chose the 5th order elliptic wave filter (EWF) [51] and the recursive part of the 4-level pipelined lattice filter (PLF) [52] as benchmarks. EWF which consists of 34 nodes, 56 edges, and 7 delays and the number of delays, $|D|$, is relatively smaller than the number of nodes, $|N|$, and the number of edges, $|E|$. On the other hand, PLF which consists of 8 nodes, 10 edges, and 8 delays and $|D|$ is comparable to $|N|$ and $|E|$.

Table 14 shows the comparison of time complexity, memory requirement, and CPU time to determine each iteration bound of EWF and PLF. In this table, NCD is the negative cycle detection method by using Bellman-Ford shortest path algorithm to detect negative cycles, LPM is the longest path matrix method, LPM' is the mixture of LPM and NCD methods by using Floyd shortest path algorithm to detect negative cycles, and MCM is the minimum cycle mean based method. The computation time of node i, $q(i)$, is assumed 1 if node i is an addition or 2 if it is a multiplication. All the CPU times are measured on a SparcStation 2 and do not include the time consumed in reading the DFG from a file.

In NCD and LPM' methods, the calculation of the iteration bound is terminated when the difference between successive guess iteration bounds becomes smaller than $1/|N|^2\gamma^2$ where $|N|$ is the number of nodes in the DFG and γ is the longest computation time of nodes [53]. While LPM and MCM derive the exact iteration bound, NCD and LPM' derive only an approximate iteration bound. Some post-calculations may be necessary to identify the exact iteration bound from the approximate.

3.2 Exhaustive Scheduling and Retiming

Time scheduling and retiming are important tools used to map behavioral descriptions of algorithms to physical realizations. These tools are used during the design of software for programmable digital signal processors (DSPs), during high-level synthesis of applications-specific integrated circuits (ASICs), and during the design of reconfigurable hardware such as field-programmable gate arrays (FPGAs). Time scheduling and retiming operate directly on a behavioral description of the algorithm, such as a data-flow graph (DFG). Since the decisions made at the algorithmic level tend to have greater impact on the design than those made at lower levels, the importance of time scheduling and retiming cannot be overstated.

Our contributions in [54] and [55] present new formulations of the time scheduling and retiming problems, and based on these formulations, new techniques are developed to determine the solutions to these problems. These formulations are valid for strongly connected (SC) graphs, where a strongly connected graph has a path $u \rightsquigarrow v$ and a path $v \rightsquigarrow u$ for every pair of nodes u, v in the graph. We focus on strongly connected graphs because these graphs traditionally present the greatest challenges when they are mapped to physical realizations due to the feedback present in the graphs.

Retiming consists of moving delays around in a DFG without changing its functionality.

As with scheduling, there is a huge body of literature on retiming, and new applications for retiming are constantly being found. For example, due to the recent demand for low-power digital circuits in portable devices, some recent work has focused on retiming for power minimization [56]. The groundbreaking paper on retiming [57] describes algorithms for tasks such as retiming to minimize the clock period and retiming to minimize the number of registers (states) in the retimed circuit. An approach to retiming which is based on circuit theory can be used to generate all retiming solutions for a DFG [58]. This approach was the motivation for our work on exhaustive scheduling. In [55], we show that retiming is a special case of scheduling, and consequently, the formulation of the scheduling problem and the techniques for exhaustively generating the scheduling solutions can also be applied to retiming.

The impact of the formulations derived in this work are as follows.

- The interaction between retiming and scheduling is important [59], and our formulations give a simple way to observe this interaction.
- We show that retiming is a special case of scheduling.
- We give solid mathematical descriptions of the scheduling and retiming problems in a common framework.
- We develop techniques for generating all solutions to a particular scheduling or retiming problem. This allows a developer the ability to search the design space for the best solution, particularly when various parameters are difficult to model and include in a cost function. This has applications to software design, ASIC design, and design for reconfigurable hardware implementations.
- Our formulations provide for a better understanding of scheduling and retiming which can be used to develop new heuristics for these problems.

The exhaustive scheduling technique is demonstrated using the fifth-order wave digital elliptic filter shown in Fig. 16. We assume that addition and multiplication require 1 and 2 units of time, respectively, and that hardware adders and multipliers are pipelined by 1 and 2

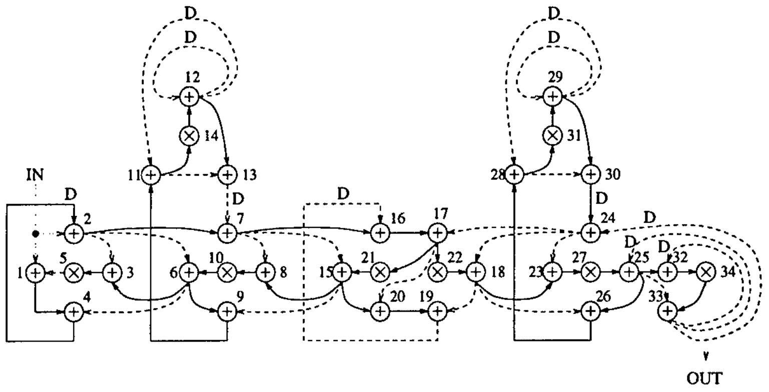


Figure 16: The fifth-order wave digital elliptic filter. The solid lines show a spanning tree used by the exhaustive scheduling algorithm.

Table 15: The results of exhaustively scheduling the filter in Fig. 16.

iter period	# sched solutions	CPU time (sec)
16	9900	0.0342
17	4669095	16.2
18	580432280	2020

stages, respectively. The results of exhaustively generating the scheduling solutions without considering resource constraints are shown in Table 15. The results of exhaustively generating the scheduling solutions which can be implemented on a given number of hardware adders and multipliers are shown on the left side of Table 16. From these tables, we can see that the time it takes to exhaustively generate only the scheduling solutions which satisfy a given set of resource constraints is orders of magnitude faster than the time it takes to exhaustively generate all scheduling solutions. The expressions in [60] can be used to compute the number of registers required by a given schedule. The results of this are shown on the right side of Table 16. Note that these results assume that internal pipelining registers cannot be shared between processors, while the results in [60] assume that internal pipelining registers can be shared between processors.

3.3 Two-Dimensional Retiming

Two-dimensional retiming [61, 62] is used to retime data-flow graphs (DFGs) which operate on two-dimensional signals such as images. As digital image processing becomes more

Table 16: The results of exhaustively scheduling the filter in Fig. 16 for a given set of resource constraints. The left part of the table considers scheduling to the minimum possible number of adders and multipliers for the given iteration period, and the right part considers scheduling to the minimum number of adders, multipliers, and registers.

iter period	resources	# sched solns	CPU time (sec)	resources	# sched solns
16	3 add, 1 mult	77	0.00288	3 add, 1 mult, 7 reg	21
17	2 add, 1 mult	98	0.0518	2 add, 1 mult, 7 reg	73
18	2 add, 1 mult	131983	11.1	2 add, 1 mult, 7 reg	40723
19	2 add, 1 mult	33948842	1700	2 add, 1 mult, 7 reg	3056246

popular in multimedia applications, the need for high speed, low area, and low power implementations of multidimensional digital signal processing (DSP) algorithms increases. Like one-dimensional retiming [57], two-dimensional retiming can be used to increase the sample rate, reduce the area, and reduce the power consumed by a synchronous circuit.

In [63], we present two techniques for retiming two-dimensional data-flow graphs (2DFGs). Each of these techniques minimizes the amount of memory required to implement the 2DFG under a clock period constraint. The first technique, called *ILP 2-D retiming*, is based on an integer linear programming (ILP) formulation which considers the 2-D retiming formulation as a whole. While this technique gives excellent results, it has slow convergence for large 2DFGs. The second technique, called *orthogonal 2-D retiming*, is formulated by breaking ILP 2-D retiming into two linear programming problems, where each problem can be solved in polynomial time. The downfall of orthogonal 2-D retiming is that the results of the two linear programming problems can sometimes be incompatible. A variation of orthogonal 2-D retiming called *integer orthogonal 2-D retiming* is also based on a linear programming formulation, and this technique solves the incompatibility problem which may be encountered using orthogonal 2-D retiming. The techniques presented in this paper result in retimed 2DFGs which require less memory than than the technique in [62] and are compatible with considerably more processing orders of the data than the technique described in [61].

Architectures

4 Discrete Wavelet Transforms

The discrete wavelet transform (DWT) has generated a great deal of interest recently due to its many applications across several disciplines, including signal processing [64], [65], [66], [67], [68]. Wavelets provide a time-scale representation of signals as an alternative to traditional time-frequency representations. Our work on wavelets includes the design of efficient DWT architectures and the development of methodologies for designing these architectures.

Several architectures for the 1-D DWT have been proposed in the past; [69] contains a survey of these architectures. For the most part, these architectures have been designed using *ad hoc* design methods because the focus has been on the architectures and not the methodologies used to design them. In our work, we are concerned with developing design methodologies which can be used to design wavelet architectures. Using these methodologies, a wavelet architecture can be designed to meet the specifications of a given application.

Our work focuses mainly on the design of folded [70] architectures for the DWT, although we also consider digit-serial [71] architectures as well. Folded DWT architectures are appealing because they lead to single-chip implementations which can be pipelined for high-throughput or low-power applications. The basic idea behind the folded architectures is to time-multiplex filtering operations performed at various rates in the algorithm description to a small number of hardware filters [72], [73], [74]. The folded hardware is clocked at the same rate as the input data, resulting in a single-rate implementation of a multirate algorithm. Detailed folded DWT architectures based on direct-form FIR filter structures were derived in [73]. Our work presents a systematic technique for constructing folded architectures for the DWT.

In the area of designing DWT architectures, our contributions are as follows:

- The development of a novel *multirate folding transformation* [75], [76] which can be used to systematically fold the multirate DWT algorithm to single-rate DSP architectures.
- The development of register minimization techniques which can be used to compute

the minimum number of registers required to implement single-rate [60] and multirate [76] DSP algorithms.

- The design of efficient lattice-based architectures for the orthonormal DWT [77], [78], [79], [76].
- A systematic technique for generating architectures for tree-structured filter banks [75].

These contributions are described in detail in the following sections.

4.1 Multirate Folding

Multirate folding [75], [76] is a technique for systematically synthesizing control circuits for single-rate architectures which implement multirate algorithms. The term *single-rate architecture* is used to describe a synchronous architecture where the entire architecture operates with the same clock period. A direct mapping of a multirate DSP algorithm to hardware would require data to move at different rates on the chip. This would require routing and synchronization of multiple clock signals on the chip. To avoid these problems, we concentrate on mapping the multirate DSP programs to single-rate VLSI architectures.

The advantages of multirate folding fall into two broad categories. The first advantage is that the multirate folding equations can be used to systematically determine the control circuitry for the architecture from a scheduled DFG. The second advantage, which is slightly more subtle, is that this formal approach can be used to address other related problems in high-level synthesis in a formal manner. Two such problems, memory minimization and retiming [57], are considered. Using the multirate folding equations, we derive expressions for the minimum number of registers required to implement the architectures, and we derive constraints for retiming the circuit such that a given schedule is valid.

The properties of multirate folding, which are described in detail in [76], are summarized below:

- Multirate folding is a novel technique for synthesizing control circuits for single-rate architectures which implement multirate DSP algorithms.

- The multirate folding equations allow us to address other problems in high-level synthesis, such as memory minimization and retiming.
- Multirate folding operates directly on the multirate DFG, avoiding the step of first constructing an equivalent single-rate algorithm description.
- Multirate folding accounts for pipelining, so architectures can be designed for high speed and low power [80] applications.
- Multirate folding is applicable to a wide variety of DSP algorithms. We demonstrate its utility by designing a discrete wavelet transform architecture in [76].

4.2 Register Minimization

In [60] and [76], expressions are derived for computing the minimum number of registers required to implement statically scheduled single-rate and multirate DSP programs.

We describe the problem using an example. After the DFG has been scheduled, specifications for the communication paths between hardware modules can be determined using systematic folding techniques [70]. Consider the multiply-add operation in Fig. 17(a), which is an algorithm DFG describing $y(n) = au(n) + v(n)$. Assume this multiply-add is part of a larger DFG which is to be implemented in hardware with an iteration period of 10, i.e., each node in the algorithm DFG will be executed by the hardware exactly once every 10 time units. If the multiply operation is executed by one-stage pipelined hardware module H_M at time units $10l + 2$, and the add operation is executed by hardware module H_A at $10l + 8$ for integer l iterations, then the connection between the multiplication and addition operations in Fig. 17(a) is mapped to the data path in Fig. 17(b). Upon examination of Fig. 17(b), one observes that at any given time, no more than one of the five delays labeled "5D" between H_M and H_A is storing a word of data that will actually be consumed by H_A. To avoid the inefficient architecture that would result from direct implementation of Fig. 17(b) in silicon, memory management is used in high-level synthesis tools to derive efficient data paths between processing modules.

Memory management consists of choosing the type of registers, number of registers, and allocation of data to these registers. The type of registers is usually dictated by the

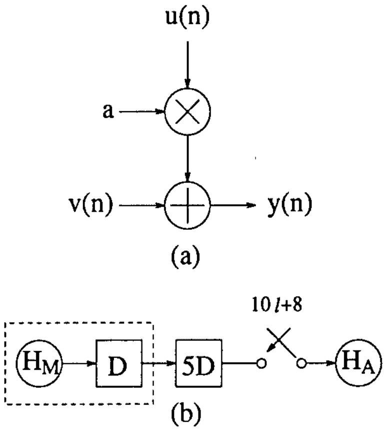


Figure 17: (a) Algorithm DFG describing $y(n) = au(n) + v(n)$. (b) Data path specification derived from the algorithm DFG for an iteration period of 10.

architecture model used. In [60], we compute the minimum number of registers required for a statically scheduled DFG under various memory models. The allocation of the data to registers is an NP-complete problem for which heuristic algorithms have been suggested [81, 82, 83].

We use life-time analysis to derive closed-form expressions for the minimum number of registers required by a statically scheduled DSP program. These techniques offer several advantages over previously used techniques. First, the closed-form expressions can be used to represent cost functions for high-level synthesis optimization tools. An example of using these closed-form expressions in an integer linear programming (ILP) formulation is given in [60]. Second, the analytical tools we introduce can be used to derive expressions for the minimum number of registers under a variety of memory models which describe how data can be allocated to memory. This is important because the target architecture may impose constraints on how data can be routed to memory. We derive expressions for three memory models, namely the *operation-constrained*, *processor-constrained*, and *unconstrained* memory models. For the unconstrained memory model, where all memory-sharing constraints are relaxed, the minimum number of registers required to implement a DFG with m nodes can be computed in $O(m^2)$ time. A third advantage of the analytical tools we introduce is that they can be used to determine memory requirements for more complex algorithm descriptions, such as DFGs which have multiplexers in the data paths.

Pipelining and retiming [57] are powerful tools used in high-level synthesis. Pipelining can be considered to be a special case of retiming. We consider an integer linear programming solution to the retiming problem, referred to as the minimum physical storage location (MPSL) retiming, which retimes a scheduled DFG such that its memory requirements are minimized under the unconstrained memory model while the schedule remains valid for the retimed DFG. We use MPSL retiming to retime a DFG which has been scheduled using the MARS design system [84], and we compare the memory requirements of MARS to a globally optimal solution. Our results show that the MARS system gives optimal or close-to-optimal results in terms of memory requirements.

The results we present can be used throughout the high-level synthesis process. Expressions for the minimum number of registers can be used during scheduling to help determine the total cost of the architecture. After scheduling, MPSL retiming can be used to optimally retime a DFG in terms of registers required for its implementation. During memory management, our techniques can be used to optimize the hardware design in terms of the number of registers required. For instance, given the scheduled DFG and the desired memory model, the minimum number of registers required can be determined, and register allocation can be performed by an appropriate register allocation scheme which guarantees completion (e.g., forward-backward register allocation [81]). Expressions for the minimum number of registers can also be used to evaluate the effectiveness of register allocation schemes which are based on heuristics, since some schemes may require more memory than the theoretical lower bound in order to maintain simple control structures.

4.3 Lattice-Based DWT Architectures

This work is concerned with the design of VLSI architectures for the orthonormal DWT which projects a signal onto the compactly supported orthonormal wavelet bases introduced in [65]. The orthonormal DWT is computed using two-channel paraunitary filter banks [85], [86]. In particular, the compactly supported wavelets which we are concerned with in this work can be computed using two-channel FIR paraunitary filter banks. These filter banks result in perfect reconstruction (PR) analysis/synthesis systems which project signals onto a set of orthonormal basis functions. Any two-channel FIR paraunitary QMF bank can be implemented using the QMF lattice [87], which has many desirable properties such as PR in the presence of coefficient quantization and low implementation complexity. These advantages of the QMF lattice motivated us to design efficient architectures for the orthonormal DWT based on this structure.

We have described folded [70] and digit-serial [88], [71], [89] architectures which are based on the QMF lattice implementation in [79]. Folded DWT architectures are appealing because they lead to single-chip implementations which can be pipelined for high-throughput or low-power applications. Digit-serial architectures also lead to single-chip implementations, and these architectures have simple interconnection and 100% hardware utilization for any number of levels of wavelet decomposition. Folded and digit-serial architectures which have been presented in the past are based on direct-form filter structures which are not as efficient as the QMF lattice for computation of the orthonormal DWT. Two contributions are made in [79]. First, we show that, for the orthonormal DWT, use of the QMF lattice structure can lead to folded and digit-serial architectures with approximately half the number of multipliers than corresponding direct-form structures, at the expense of an increase in the system latency. Furthermore, these architectures possess better finite word-length properties. Second, we present techniques for mapping the 1-D orthonormal DWT to folded and digit-serial architectures which are based on the QMF lattice structure.

The basic idea behind the folded DWT architectures is to time-multiplex filtering operations performed at various rates in the algorithm description to a small number of hardware filters [72], [73], [74]. The folded hardware is clocked at the same rate as the input data, resulting in a single-rate implementation of a multirate algorithm. Detailed folded DWT architectures based on direct-form FIR filter structures were derived in [73]. In [79] and [77], we present a systematic algorithm to construct folded architectures based on the QMF lattice for the orthonormal DWT. A detailed example is given to demonstrate the algorithm, and comparisons are made with the folded direct-form architectures in [73].

DWT architectures based on digit-serial processing techniques [88], [71], [89] were introduced in [73]. The number of bits processed per cycle, called the *digit-size*, varies throughout the digit-serial DWT architecture. The digit-size is chosen such that the architecture is single-rate and achieves 100% hardware utilization. This is in contrast to folded DWT architectures, which result in less than 100% hardware utilization. It may be noted that while it may be possible to design folded DWT architectures which achieve 100% hardware utilization, the control complexity in these architectures would be much higher. In [73], digit-serial architectures were presented for direct-form implementations of the DWT. In [79] and [77], we present a general method based on polyphase decomposition of filters [85] for implementing two-channel systems using digit-serial processing techniques. This method is used to derive digit-serial architectures based on the QMF lattice for the orthonormal DWT.

4.4 Architectures for Tree-Structured Filter Banks

In this paper, we develop a methodology for designing efficient VLSI architectures for M-ary tree-structured filter banks which are constructed from a single M-channel FIR filter bank. Full and pruned tree-structured filter banks are useful for many DSP applications, such as signal coding and analysis. Recent interest in the discrete wavelet transform (DWT) has significantly increased the number of applications for tree-structured filter banks because the DWT can be computed using a pruned tree-structured filter bank [64], [65]. Computation of wavelet packet bases is another application of pruned tree-structured filter banks [90]. For full and pruned tree-structured filter banks, FIR filters are used almost exclusively in practice because excellent M-channel FIR filter banks can be designed without worrying about the implementation issues associated with IIR filter banks. For this reason, we concentrate on designing architectures based on M-channel FIR filter banks.

Synthesis of folded DWT architectures was accomplished in [73] by scheduling the filtering operations to hardware and then synthesizing the control using life-time analysis and forward-backward register allocation. Orthonormal DWT architectures based on the QMF lattice structure were developed in [77] by iteratively applying single-rate folding techniques [70]. The design methodology we have developed in [75] operates directly on the multirate algorithm description of M-ary tree-structured filter banks, including the DWT. This methodology simultaneously schedules and retimes the system to maintain low control complexity and low memory requirements in the synthesized architecture. The methodology has several attractive features.

- The methodology can be used to synthesize architectures for a wide class of multirate DSP algorithms while previous techniques were restricted to handle only synthesis of DWT architectures.
- The methodology is simple because our scheduling algorithm and folding equations operate directly on the multirate algorithm description rather than first constructing an equivalent single-rate algorithm description.
- The resulting architectures have simple control and low memory requirements.
- The methodology accounts for pipelining so architectures can be designed for high-throughput and low power [80] applications.
- The methodology provides a complete high-level description of architectures for any uniform implementation style, i.e., architectures which are bit-parallel, bit-serial, or digit-serial with fixed digit-size.

5 High-Speed Digital Communications: HDSL/ADSL/VDSL

In recent years, a consensus has been growing that the use of integrated digital network carrying all kinds of information (speech, computer data, video, medical imaging, etc.) is imminent in the future. The right starting point to achieve this dream was to utilize the under-loaded telephony network for data communication. More efficient utilization of the telephony network, also known as the subscriber loop plant, for high speed digital communication has been stirring much interest recently. The trends towards integrated networks and the investments involved make even a small improvement in the performance/cost tradeoff a worthwhile step. Using twisted pair telephone loops to transmit high speed data is a cost driven choice. This led to the introduction of the digital subscriber loop (DSL) as a way to denote transmitting digital information over subscriber loop plant. Fig. 18 shows the major two steps when connecting the central office to the end user. One main cable that has many subscriber loops in it connects the central office to the local distribution box, residing on the curb side. Many connections go from the distribution box to different end users. Due to

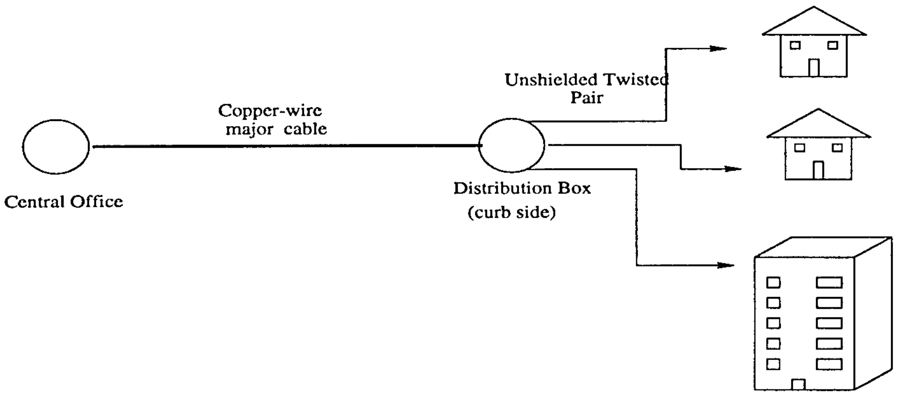


Figure 18: The Subscriber Loop Plant

the replacement costs, the curb to the user connection is the most cost sensitive connection in the data communication link. This connection is also called the premises environment. Numerous high speed digital communication systems have been proposed for the digital subscriber loops. The High-speed Digital Subscriber Loop (HDSL) [91], was introduced as a step further from the DSL. As the video on-demand was a driving force for this investigation, there was no need for equal transmission rates in each direction of the communication link. Instead, more rate was given to the central office to the user direction on the expense of the user to the central office direction. That kind of topology was introduced as the Asymmetric Digital Subscriber Loop (ADSL) and Very High-speed Digital Subscriber Loop (VHDSL) [92][93][94]. Many systems have been proposed to maximize the system performance for any given channel. The performance is measured by the baud rate and the bit error rate at the receiver end.

The Discrete Multitone (DMT) [95] [96] [97] and the Carrierless Amplitude/Phase modulation (CAP) [98] [99] are two viable techniques for high speed digital transmission over copper wires. The Discrete Multitone system DMT, was recently introduced as a practical implementation of the known multitone channel. The CAP system is a QAM like modulation that was introduced in the seventies. The ordinary CAP scheme is a 2-dimensional modulation with Hilbert-pair signaling.

In our research, the idea of expanding the CAP system beyond 2-D was investigated. This idea offers many potentials for improvement over the original CAP system. One possible advantage can be increasing the system throughput on the expense of the increased complexity and increased receiver interference energy. A 50% increase in the system throughput is possible by going from two-dimensional to three-dimensional signaling. Another possible application is in the area of the multiple access communication environment for the premises environment. This would allow multiple users to enjoy their own channels of communications while they use the same physical communication link. We call this new technique Orthogonality Division Multiple Access (ODMA).

The summary of the two techniques, DMT and CAP, is introduced in section 5.1. As our own investigation proved that the CAP system is more promising in terms of the performance cost tradeoff, the CAP system will be given more emphasis here. The feasibility of constructing 3-D signaling for the CAP and writing the problem in the form of an optimization problem is explained in section 5.2. Sequential Quadratic Programming was used to solve the optimization problem as a Minimax problem. The condition for perfect reconstruction (PR) with the 3-dimensional case is studied. The performance of the 3-dimensional system is tested with simulations for the unshielded twisted pair copper wires. A summary of the simulation results is introduced in section 5.4. The possibility of having higher dimensions to allow the multiple access option is introduced in section 5.3. This multiple access option allowed with higher dimensions for the CAP system will be suited for the premises environment.

The Least Mean Square (LMS) adaptive algorithm is used in implementing the CAP receiver equalization, as it offers good performance with reasonable complexity. Pipelining the LMS using a moving average was previously developed [100]. As the performance of the moving average pipelining starts to degrade with larger LMS filters, a new IIR based relaxation for the LMS is introduced in section 5.5. The summary of this report as well as the directions for possible future work are explained in section 5.6.

5.1 Background

5.1.1 Motivation

The problem of the DSL brought many interesting challenges in digital communications. Technically speaking, transmitting huge amounts of digital information over integrated network has many possible solutions. The feasibility of each solution differs according to the specification needed and the amount of money that can be invested. First answer can be simply replacing the whole current network with fiber optic links. This solution is known as Fiber To The Home (FTTH). As FTTH involves unacceptable replacement costs, this solution is beyond the foreseen future [101]. Another possible scenario is to replace only the link between the central office and the distribution box. The link from the distribution box to the end user is left, as it is the most cost effective one. This solution is known as Fiber To The Curb (FTTC), and the bottleneck connection from the curb to the end user is called the premises environment. The FTTC has many advantages as it can support the IMTV [101] for multiple users per link, and it can be upgraded in the future FTTH. Another possible solution can be employed by leaving the copper wire links intact and trying to maximize the transmission throughput. Although it has minimal replacement costs, this kind of communication link has severe limitation if compared to the FTTC and FTTH scenarios. The rates that need to be supported by this link are the T1 rate of 1.5 MHz and the DS1 rate of 6.1 MHz.

5.1.2 Discrete Multitone

Most of the realistic channels have non-flat characteristics, which require a more sophisticated coding scheme to get closer to the theoretical limit of the channel capacity. In theory, one can achieve the channel capacity limit by transmitting a signal that has the same spectral shape of the channel. The theoretical limit for the channel capacity can be achieved with the water pouring solution [96]. One way to approximate the water pouring solution is by using the Discrete Multitone technique. With the DMT approach, the channel is approximated as a finite number of piecewise continuous subchannels, each with a flat characteristic. Each subchannel is then modulated separately with a QAM carrier which will generate a set of complex QAM symbols. The symbols generated are appended with their hermitian extension and passed through an IFFT block to get a sequence of real samples. At the receiver, an FFT block is used to retain the original set of symbols. The channel distortions including noise and interference will alter the value of the received symbols. For each subchannel, a separate equalizer is used to invert the effect of the channel and to suppress the distortion

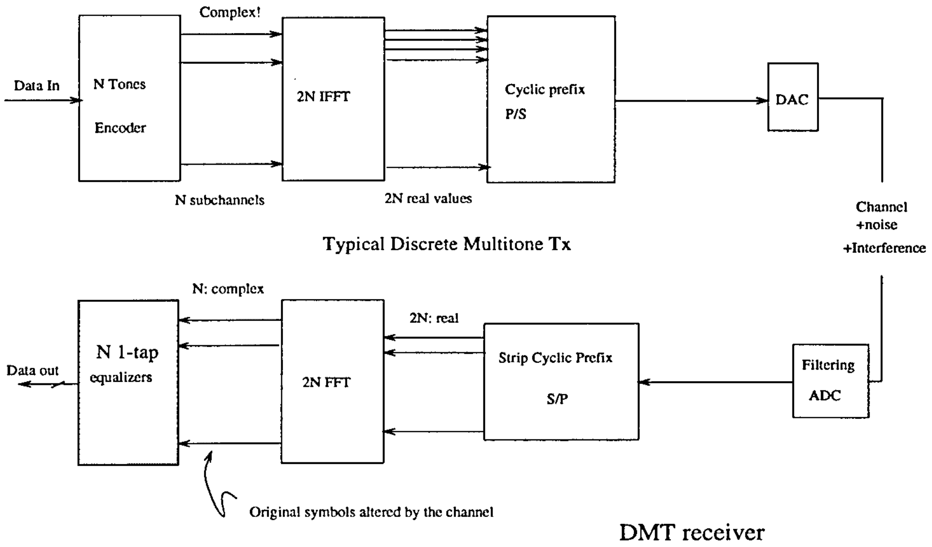

Complex!
Data In
N Tones
Encoder
2N IFFT
Cyclic prefix
P/S
DAC
N subchannels
2N real values
Channel
+noise
+Interference
Typical Discrete Multitone Tx
N: complex
2N: real
N 1-tap
equalizers
Data out
2N FFT
Strip Cyclic Prefix
S/P
Filtering
ADC
Original symbols altered by the channel
DMT receiver

Figure 19: The DMT Transceiver

Table 17: Finite word-length selection for ADSL: DMT

No. of bits	output error in dB
double prec.	-31
12	-30
8	-27
4	-18

added to the signal. Single-tap linear equalizers were previously suggested for such a scheme. Another post-channel equalizer is still needed to limit the interblock interference introduced by the channel memory. The full structure of the DMT transceiver is shown in Fig. 19.

The blocks within the receiver are generally easy to implement for high speed applications, except for the equalization part. Pipelining techniques can be employed easily to achieve very high speed architectures for the FFT block. As only half the output of the FFT block is used, the FFT block can have reduced complexity by eliminating all redundant hardware. Using a bank of 1-tap linear equalizer would achieve acceptable performance with reasonable complexity. The effect of having finite word length was studied for implementing the DMT receiver. An 8-bit word length was found to give acceptable performance and allowed a VLSI implementation of the receiver equalizer (Table 1).

5.1.3 Carrierless AM/PM Transceiver

Carrierless AM/PM (CAP) is a bandwidth efficient, 2-dimensional passband transmission scheme. The basic idea of the CAP system is to use two signals as signature waveforms to modulate two data streams. The bandwidth efficiency is achieved in two steps. The first step is by multilevel encoding of the data stream. Using 4-level encoding for each dimension will generate the so-called CAP-16. Fig. 20 shows the 2-dimensional signal constellation for the CAP-16. The other step for achieving bandwidth efficiency is using efficient signature waveforms. The theoretical limit of that parameter is achieved when using Nyquist signaling. Efficient shaping necessitates using signature signals that occupy more than one symbol period in time.

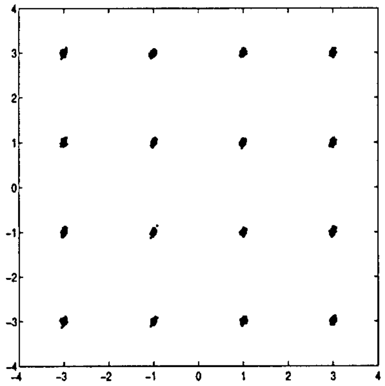
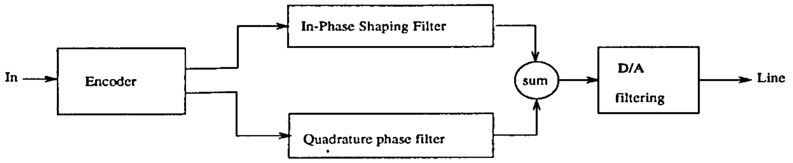


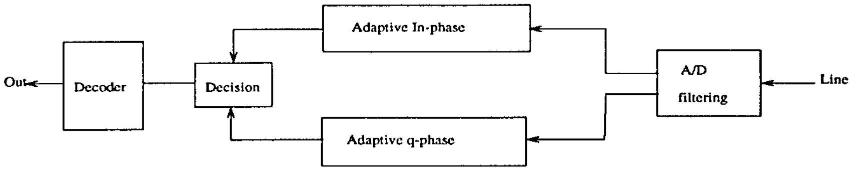
Figure 20: 16-point signal constellation for CAP-16

Extending the signature waveform will shrink the frequency domain characteristics of the signal. This extension of signature waveforms will lead to overlapping signatures of successive symbols. The design of the signature waveforms should ensure no intersymbol interference between consecutive symbols, and also no crosstalk between symbols in each dimension. In practice, different signals can be used to meet those criteria. Examples for that are the raised cosine signal and the square root raised cosine signal.

The advantage of going to 2-dimensional signaling is to be able to retain the same bandwidth efficiency for a passband signal. The two orthogonal signals used as signature wave-


In
Encoder
In-Phase Shaping Filter
Quadrature phase filter
sum
D/A
filtering
Line

CAP Tx


Adaptive In-phase
Out
Decoder
Decision
A/D
filtering
Line
Adaptive q-phase

CAP Rx

Figure 21: CAP System Transmitter and Receiver

forms are modulated versions of an original baseband signal, and are given by

$$\begin{aligned} f_1(t) &= g(t)\cos 2\pi f_c t \\ f_2(t) &= g(t)\sin 2\pi f_c t, \end{aligned} \tag{44}$$

where $g(t)$ is the baseband signal and f_c is a frequency that is larger than the largest frequency in $g(t)$. The pair $\{f_1, f_2\}$ is called Hilbert pair. Fig. 22 shows the time domain modulated raised cosine signature waveforms and the normalized frequency characteristics for symbol rate of 25MHz.

The structure of the CAP transceiver is shown in Fig. 21. The data stream is scrambled into two symbol streams, and each is modulated with the corresponding signature waveform. The receiver is implemented in adaptive fashion to invert both the channel and the signature filters and retrieve the original sequence of symbols. Many topologies can be used to implement the receiver such as the linear equalizer and the decision feedback equalizer. The major challenge in designing the receiver is to guarantee perfect reconstruction (PR) of the original sequences. The transmitter, as well as the receiver, are implemented in a digital fashion. The transmitter signature filters are implemented as fixed finite impulse response (FIR) filters. To implement the system with that topology, the sampling rate of the implementation must be high enough to prevent aliasing effects. For that implementation, the input symbol

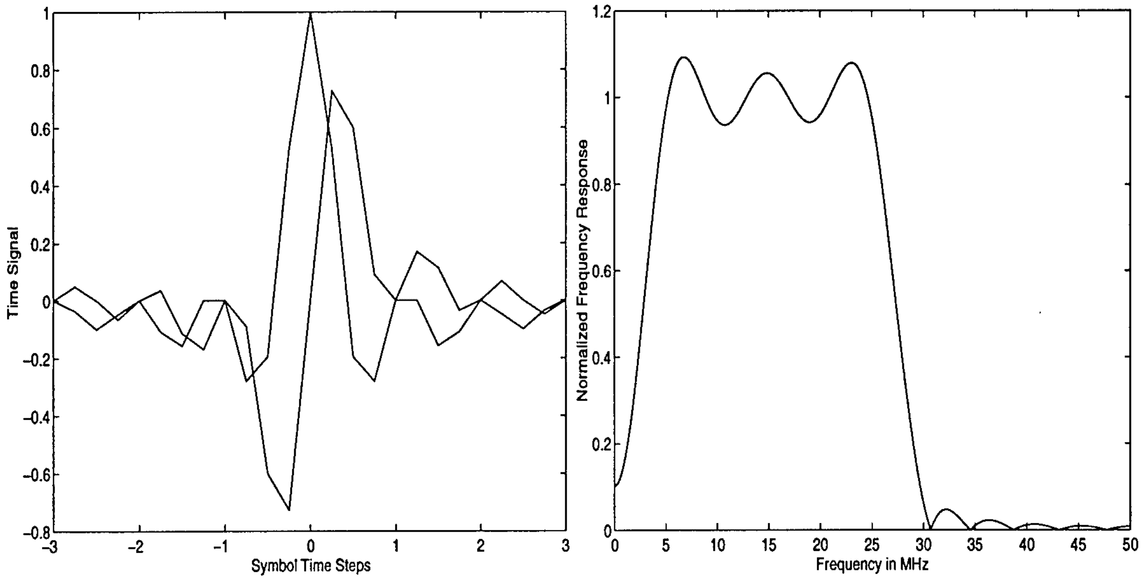


Figure 22: Two Signature Waveforms Over a Span of 6T in Time and Frequency

sequences are up-sampled (usually by a factor of 4 or 5) to match the implementation sample rate. The performance of the CAP-16 system is found to be acceptable in terms of system throughput and receiver bit error rate for the unshielded twisted pair environments.

5.2 Three-Dimensional CAP

One option to increase the CAP system throughput is by increasing the number of levels in the multilevel encoding. This means going to larger signal constellation sizes; CAP-32, CAP-64, CAP-128 etc. Another idea is by using higher dimension signaling. The idea is based on modulating the data streams using more than two signature waveforms. The major obstacle in designing the signals used as signature waveforms is the orthogonality over multiple symbol periods.

Examining the original 2-dimensional CAP system as shown in detail in Fig. 23 shows it is a multirate transmultiplexer problem. The channel and the interferences are taken out from the figure to emphasize the problem we are solving. $\{s_0, s_1, s_2\}$ are the three input sequences, scrambled from the input bit stream. After going through the multirate transmultiplexer, the output sequences, $\{\hat{s_0}, \hat{s_1}, \hat{s_2}\}$, are needed to be as close as possible to the input sequences.

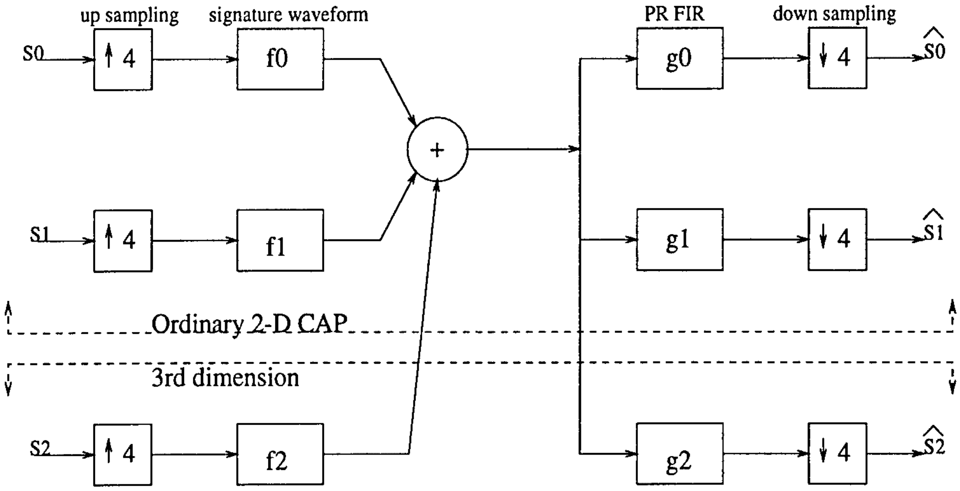


Figure 23: Expanding the 2-D CAP into 3-D CAP

The receiver equalizer solves the problem to find the perfect reconstruction (PR) solution. The transmultiplexer has the multiple input multiple output transfer matrix T [102]

$$T = G\Gamma H, \tag{45}$$

where G, and H are the polyphase phase decomposition matrices for the receiver and the transmitter, respectively, and Γ is a permutation matrix that depends on the different number of delays inserted in the system filters. It can be easily shown that PR at the receiver end can be achieved if and only if

$$T = z^{-n}I, \tag{46}$$

where I is the identity matrix and z^{-n} denotes n delay elements. For proper system design, the receiver must be implemented with FIR topology. If this is not met, the adaptive equalizer at the receiver will be an IIR adaptive filter, which would not be tractable. In this article, the PR condition will always be assumed to have FIR receiver topology. The minimax optimization algorithm was performed to find three signals that can be plugged into a PR system with FIR receiver topology. The optimization used was based on Sequential Quadratic Programming method [103]. The optimization problem is stated mathematically as finding the set $\{f_0, f_1, f_2\}$, that solves

$$\min_{\{f_0,f_1,f_2\}} max(|F - R|)$$
$$\text{s.t. } G\Gamma H = z^{-n} I,$$

where F is the frequency characteristics of the signals set $\{f_0, f_1, f_2\}$, R is the passband frequency response of the raised cosine pair, and G is found by inverting H to obtain the polyphase decomposition of the receiver filters. The three signals found using the minimax optimization approach are plotted in Fig. 24. The advantage of having three dimensions over the 2-D CAP is to allow 50% increase in throughput on the expense of increased system complexity and 2 to 3 dB of receiver error.

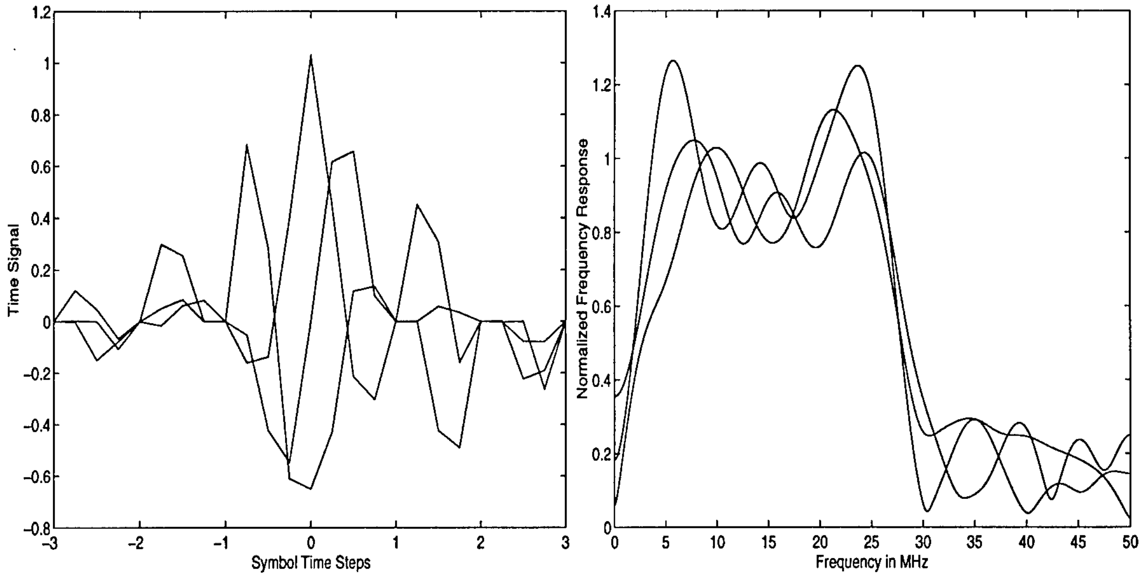


Figure 24: Solution of Optimization Problem: 3 signals in Time and Frequency Domains

5.3 ODMA

The concept of three-dimensional CAP can be extended to more dimensions, opening the door for multiple access option. As the signals generated using that scheme are orthogonal in nature, we call this technique Orthogonality Division Multiple Access (ODMA). Fig. 25 shows the possible structure for ODMA. Simulations were carried out to test the feasibility of ODMA. Signals were generated using the same optimization problem defined in the previous section. If we have the overall symbol rate to be constant $1/T$, We can allow the transmission to take place using K orthogonal signals. The upsampling of each stream is kept at $2K$. This means we don't have any increase in the system throughput, but rather we have the multiple access option. Simulations were carried out to generate 4-D and 6-D signals. Fig. 26 along with Fig. 22 give the signals designed for 4-D ODMA. The signals were designed for upsampling of 8, and are to carry the same 2-D symbol rate. The limit on the number of signals that can be generated still needs more investigation.

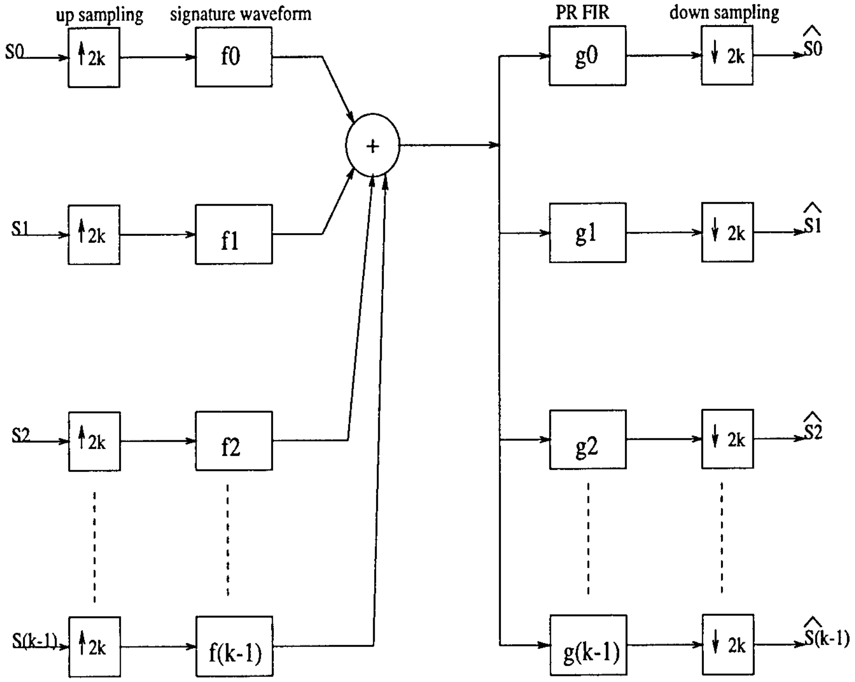


Figure 25: ODMA

5.4 Simulation Results

Simulations were carried out to test the functionality of the proposed 3-D CAP system over the UTP copper wire. The channel was inserted in the transmultiplexer problem, and the receiver PR is performed with linear adaptive equalizer. Using the EIA/TIA-568 standard for unshielded twisted pairs of categories 3 and 5, the PR condition was met using the adaptive linear equalizer. Fig. 27 shows the 3-dimensional signal constellation at the receiver end for the category-3 cable case with no near end crosstalk (NEXT). Similar performance is achieved for category-5 cables. Fig. 28 shows the signal constellation for the category-5 case

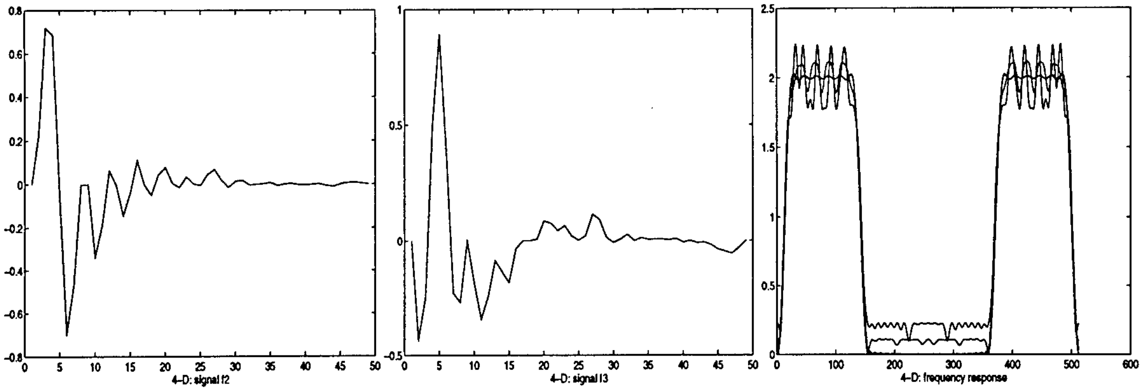


Figure 26: 4-D, in time and frequency

in the presence of NEXT. The adaptive linear equalizer achieves PR at the receiver and the adaptation rule used is the LMS algorithm.

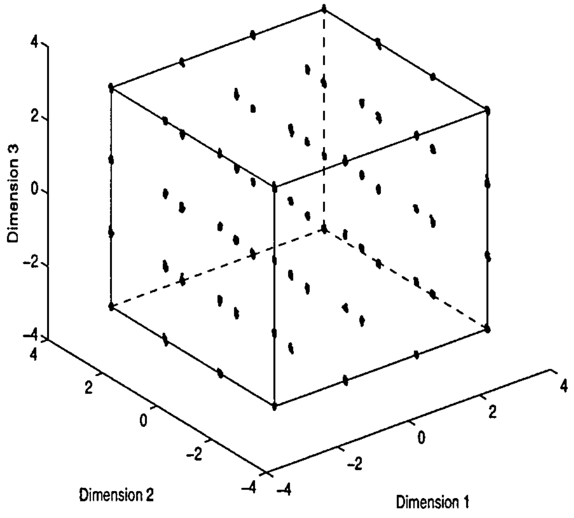


Figure 27: Three Dimension Constellation

5.5 LMS Relaxation

Pipelining is a major technique for developing high speed digital signal processing (DSP) architectures. The pipelining offers an increase in the sampling rate by reducing the critical path propagation delay. Pipelining of adaptive filters is made difficult due to the coefficient

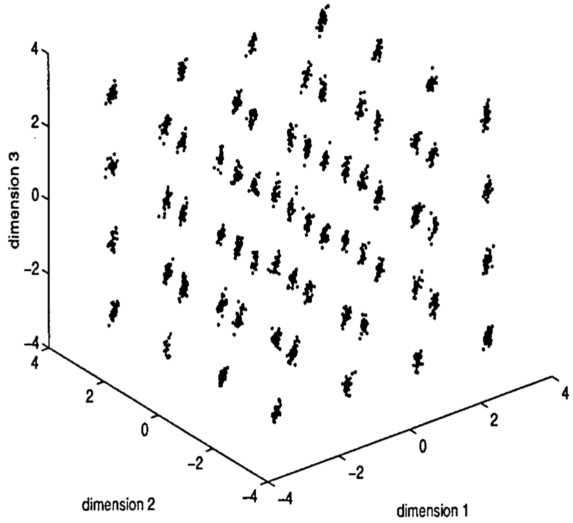


Figure 28: PR for Category-5 UTP, with NEXT

update loop. Keeping the input-output relation of the serial LMS while inserting lookahead delays in the architecture closed loops introduces a very expensive hardware overhead.

The relaxed lookahead LMS is an approximation of the LMS that can be pipelined. It is obtained by relaxing the constraint of maintaining the exact input-output mapping. The input-output relation is maintained only in the stochastic sense. To introduce D_1 delays in the outer loop and D_2 delays in the inner loop, the input-output relation becomes very complicated. The approximation introduced in [100] will lead to the following simple filter equations;

$$\begin{aligned} W(n) &= W(n-D_2) + Z(n) \\ Z(n) &= \mu \sum_{i=0}^{LA-1} e(n-D_1-i)U(n-D_1-i) \\ e(n) &= d(n) - W^T(n-D_2)U(n), \end{aligned} \tag{47}$$

where $W^T(n) = [w_1(n), w_2(n), ...w_N(n)]$ is the filter tap weights, U(n) is the input vector, and Z(n) is the feedback weight update variable . LA is called the lookahead factor, and we should keep $LA \leq D_2$. The relaxed LMS is essentially the serial LMS with delays inserted in the closed loops and a moving average block is added to compensate for the performance degradation of the filter. Fig. 29 shows the closed loop of only one tap with the delays inserted. This approximation introduces added error to the serial LMS misadjustment. The misadjustment for small LMS update factor μ and large N can be written for a normalized power environment as;

$$M = \frac{\alpha N\mu}{2-(\alpha N\mu)}, \tag{48}$$

where α is a factor determined by the input sequence eigen-structure. The misadjustment can be expanded with higher powers of μ neglected as;

$$M = \frac{\alpha N\mu}{2}(1+\frac{\alpha N\mu}{2}). \tag{49}$$

It is apparent from this equation that the relative misadjustment increases with the filter order, N, as illustrated in the previous equation.

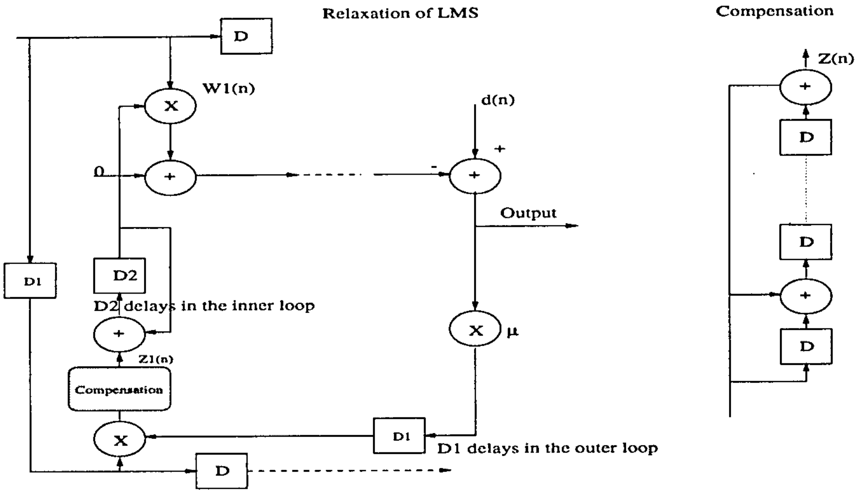

Relaxation of LMS
Compensation
D
W1(n)
X
d(n)
+
0
+
-
+
Output
Z(n)
+
D
D
+
D
D1
D2
D2 delays in the inner loop
+
Z1(n)
Compensation
X
μ
X
D1
D1 delays in the outer loop
D

Figure 29: Relaxed Lookahead LMS

Another approximation is introduced by replacing the MA compensation with a fixed one-pole IIR filter as shown in Fig. 30. The system equations will be:

$$\begin{aligned} W(n) &= W(n-D_2)+Z(n) \\ Z(n) &= aZ(n-1)+e(n-D_1)U(n-D_1) \\ e(n) &= d(n)-W^T(n-D_2)U(n), \end{aligned} \tag{50}$$

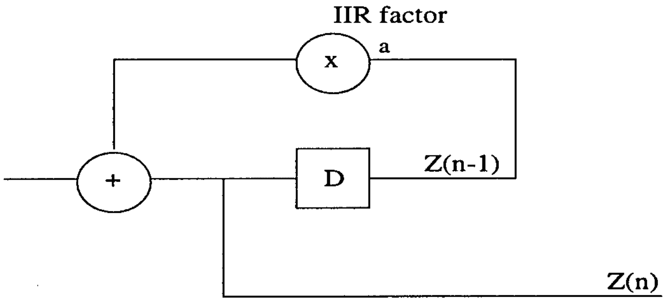


Figure 30: IIR Compensation

where "a" is the IIR fixed coefficient. The compensation block overhead can be significantly reduced if the the value of "a" is restricted to 2^{-k}, for integer values of k. The implementation of the coefficient will be a simple shift operation. The IIR relaxation can offer up to 3dB improvement over the relaxed lookahead technique.

5.6 Concluding Remarks

During the course of this project, a set of new tools were developed to facilitate the design and implementation of line equalization for the HDSL/ADSL/VDSL applications. IIR based relaxation for the LMS was found to have better performance as compared to the relaxed LMS. This improvement increases for large LMS filters. The DMT receiver was implemented for finite word length, and with significant reduction in hardware costs. The CAP system was studied for multi-dimensional signaling. Expanding the ordinary 2-dimensional CAP into higher dimensions offers a practical solution for throughput increase without the need to increase the number of levels in the multilevel encoding. The PR condition with FIR receiver topology was achieved by finding the suitable signals for the transmitter. Minimax optimization proved to be a convenient tool for designing the required signals. Expanding the system into even higher dimension looks possible, but the problem still needs more investigation. Although the final bit error rate of the 3-D system is few dB worse than the 2-D system when using linear equalizer, the overall system performance remains acceptable for the UTP environment. The advantage of not using more levels of the encoded signal becomes apparent when implementing the receiver equalizer. Increasing the number of levels per dimension makes it more difficult for the equalizer to identify each level. Another frontier that is opened by the multi-dimensional CAP is the ODMA. More work is still needed to investigate the full potential of the ODMA.

6 Finite Field Arithmetic and Reed-Solomon Coders

Finite field arithmetic operations have received a lot of attention because of their important and practical applications in cryptography, coding theory, switching theory, and digital signal processing. The finite field $GF(2^m)$ has 2^m elements and each of them is represented by m binary digits based on the primitive polynomial $f(x)$. For such representation, addition and subtraction are bit-independent and relatively straight-forward. However, multiplication, exponentiation and division are much more involved. Hence design of efficient architectures to perform these arithmetic operations is of great practical concern. In this project, several novel architectures on finite field multiplication and exponentiation have been derived and their advantages have been compared with some existing architectures.

Reed-Solomon codes are the most frequently used error control codes with applications ranging from digital audio disc players to the spacecraft. Its encoding and decoding process makes use of finite field arithmetic. Therefore, based on one of the proposed efficient multiplication algorithms, an efficient Reed-Solomon encoder has been derived during this project.

Four papers were published in this area. In this report, the main results are summarized in the order of their publication time as following:

1. Efficient power based Galois Field arithmetic architectures [104].
2. Low latency standard basis $GF(2^m)$ multiplier and squarer architectures [105].
3. Efficient standard basis Reed-Solomon encoder [106].
4. Efficient finite field serial/parallel multiplication [107].

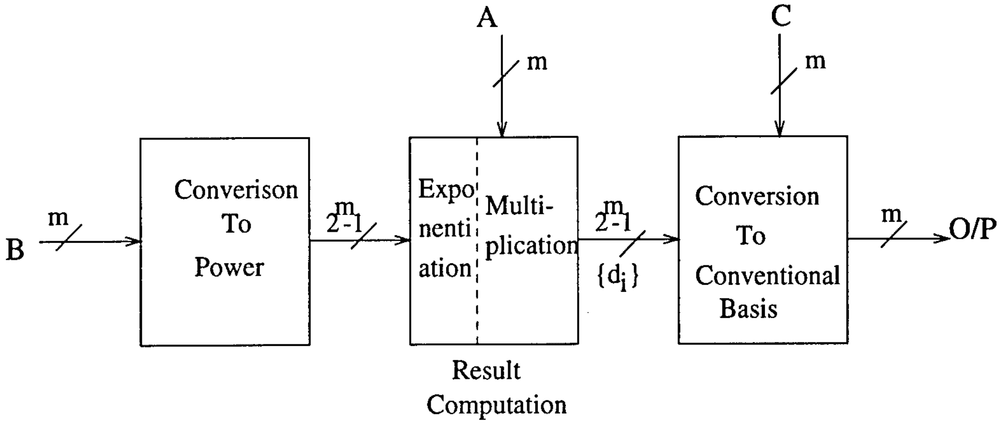


Figure 31: **System Level diagram for Proposed New Architecture**

6.1 Efficient Power Based Galois Field Arithmetic Architectures

The concept of representing the finite field elements in terms of the primitive element α has been utilized to derive a new architecture to perform a general operation like $AB^n + C$ [104]. Once the elements are expressed in terms of the primitive element α, the power of the result can be computed, i.e., the power need to be added for multiplication, subtracted for division and multiplied for exponentiation. After that, the power of the result can be converted to conventional basis representation. Fig. 31 shows the system level diagram of the proposed architecture.

6.1.1 Conversion to Power

In the conventional basis representation, each element of GF(2^m) can be represented as a sum of $1, \alpha, \alpha^2, ...\alpha^{m-1}$. If we use 2^{m-1} bits to represent power, i.e., each bit represents a particular power, we can get the power of a particular operand by a logical AND of m i/p variables $\alpha^0, \alpha^1, \alpha^2, \ldots, \alpha^{m-1}$. In this architecture, the operand B is converted into the power form while the operand A is left in the conventional basis. The output of this block has $2^m - 1$ bits with each bit corresponding to a power of α.

Example. $B = \alpha^3 + \alpha = \alpha^9$, at the output of this block, the bit corresponding to α^9 is set to 1 while the rest of bits are all set to 0.

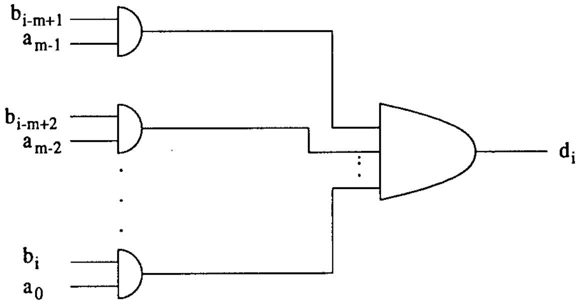


Figure 32: **Result Computation Module**

6.1.2 Result Computation

Given an element in its power form, exponentiation B^n is equivalent to multiplying the power and computing the result mod $2^m - 1$. This computation can be done *apriori* provided n is known.

For multiplication AB^n, let

$$\begin{aligned} B^n &= b_p\alpha^p \\ A &= a_0 + a_1\alpha + ... + a_{m-1}\alpha^{m-1} \end{aligned} \tag{51}$$

where $0 \leq p < 2^m - 1$. Then,

$$AB^n = a_0b_p\alpha^p + a_1b_p\alpha^{p+1} + ... + a_{m-1}b_p\alpha^{p+m-1}. \tag{52}$$

The architecture to perform this operation is shown in Fig. 32. Notice that in general to perform summation in $GF(2)$ we need an XOR gate, but in this case for each α^i, at most only one of the contributing terms will be 1 and therefore the XOR can be replaced by an OR gate. Thus, for each power bit, we need m (2 input) AND and $m-1$ (2 input) OR gates. The $\{d_i\}$'s for $0 \leq i < 2^m - 1$ are computed using $2^m - 1$ circuits similar to Fig. 32 and these are inputs to the conversion unit to convert the result AB^n from power to conventional basis. This result can then be added to C as shown in Fig. 31.

Example. (Cont') In our Example, $b_6 = 1$. Thus, only terms d_6 through d_9 could be 1. Rest of d_i's are 0. Also, $A = \alpha^2 + \alpha$, therefore, $a_2 = a_1 = 1$ and $a_3 = a_0 = 0$. Computing d_6 through d_9, we get

$$d_6 = 0.0 + 1.0 + 1.0 + 0.0 = 0$$

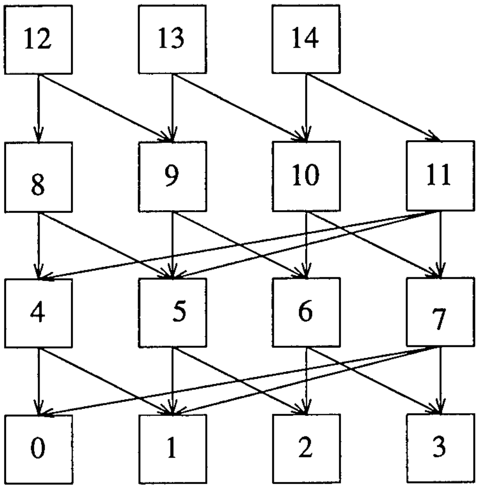


Figure 33: **New Encoder Architecture**

$$\begin{aligned} d_7 &= 0.0 + 1.0 + 1.1 + 0.0 = 1 \\ d_8 &= 0.0 + 1.1 + 1.0 + 0.0 = 1 \\ d_9 &= 0.1 + 1.0 + 1.0 + 0.0 = 0. \end{aligned}$$

6.1.3 Conversion to Conventional Basis

To convert from power to conventional basis, we can utilize a 2^m *to* m encoder. This would, however, require m XOR gates each with 2^{m-1} inputs. This exponential dependence of the number of i/ps to a gate on m is clearly not a desirable property from VLSI implementation viewpoint.

Utilizing the property of Galois Field $GF(2^m)$ that each element can be represented as a sum of $1, \alpha, \alpha^2, \ldots, \alpha^{m-1}$, it is, however, possible to tradeoff the number of inputs with the number of gates required for encoding. This new encoder architecture is illustrated in Fig. 33 for GF(2^4) generated by $\alpha^4 = \alpha + 1$. In Fig. 33, each of the box marked i receives d_i from the result computation module as an input and performs XOR operation of this and other inputs associated with this box. After 3 delays, the element's representation will be available in the conventional basis. In general, such an architecture for the encoder will need $O(2^m)$ XOR gates while the 2^m to m encoder needs $m(2^{m-1} - 1)$ gates [108].

Table 18: **Summary of Hardware Requirements**

Item/Operation	**Massey-Omura [109]**	**Architecture of [110]**	**New Architecture**
Basic Cell	$O(0.5m^2)$ AND , $O(0.5m^2)$ XOR	m^2 AND m(m+k) XOR	2m-1 AND , $m-1$ OR, 1-2 XOR
AB^n	m^2 copies	2m-1 copies	$2^m - 1$ copies
Latency	m+1	$2m^2 + 3m$	O(m)
Time step	O($\lceil log_2 0.5m^2 \rceil$)XOR, 1 AND	AND, XOR	$O(m - \lceil log_2 m \rceil)$ XOR

6.1.4 Comparison with Other Architectures

The proposed architecture has been compared with the Massey-Omura architecture [109] and the exponentiation architecture presented in [110]. The comparison will be in terms of 2 input gates.

In general, for GF(2^m), the Massey-Omura architecture requires $m^2 O(0.5m^2)$ AND and XOR gates. The latency is $m + 1$ time steps where each time step has the delay of O($\lceil log_2 0.5m^2 \rceil$)XOR and 1 AND gate.

The architecture presented in [110] requires $2m - 1$ multipliers where each multiplier needs m^2 AND and $m(m + k)$ XOR gates where k is the number of non-zero terms in the primitive irreducible polynomial used to generate GF(2^m). The latency is $2m^2 + m$ time steps where each time step is the delay of AND followed by a XOR gate.

Our architecture requires $(2m - 1)(2^m - 1)$ AND, $(m - 1)(2^m - 1)$ OR and $O(2^m - 1)$ XOR gates. The latency is also $O(m)$ time steps where each time step is the delay of $O(m - \lceil log_2 m \rceil)$ XOR gates.

These results are summarized in Table 18. We consider $GF(2^m)$ generated by a primitive irreducible polynomial with k non-zero terms. The throughput rate for all the architectures is 1 result every clock cycle.

It is apparent from Table 18 that the proposed new architecture has a low latency. It is also hardware efficient for $m \leq 6$. For larger finite fields, the architecture proposed in [110] is more hardware efficient. The architecture proposed in [111] is based on a *square and multiply* algorithm for exponentiation. It utilizes parallel-in-parallel-out multipliers based on a standard basis representation. It was shown in [112] that in general the standard basis multipliers have lower design complexity and are easier to extend to large finite fields because

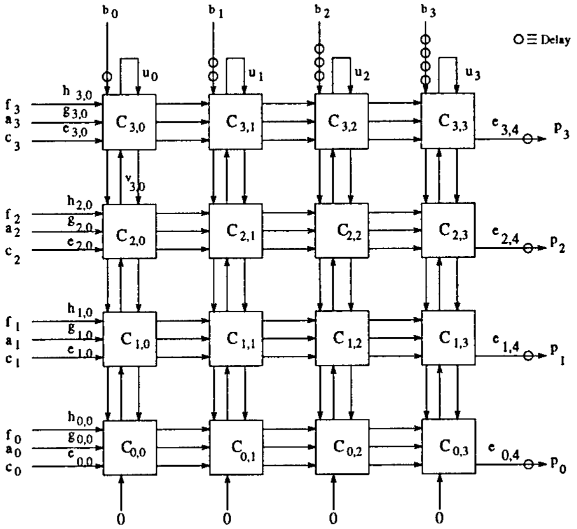

b0
b1
b2
b3
○ ≡ Delay
u0
u1
u2
u3
h 3,0
g3,0
e3,0
f3
a3
c3
C3,0
C3,1
C3,2
C3,3
e3,4
p3
v3,0
h2,0
g2,0
e2,0
f2
a2
c2
C2,0
C2,1
C2,2
C2,3
e2,4
p2
h1,0
g1,0
e1,0
f1
a1
c1
C1,0
C1,1
C1,2
C1,3
e1,4
p1
h0,0
g0,0
e0,0
f0
a0
c0
C0,0
C0,1
C0,2
C0,3
e 0,4
p0
0
0
0
0

Figure 34: **Parallel-in-Parallel-out $GF(2^4)$ Multiplier**

of their simplicity, modularity and regularity in architecture. It is also easier for architectures based on standard basis representations to allow programmable primitive irreducible polynomials, thus providing the user with greater flexibility in system design.

6.2 Low Latency Standard Basis $GF(2^m)$ Multiplier and Squarer Architectures

6.2.1 Parallel-in-Parallel-out Multiplier

6.2.1.1 Multiplier Architecture

A low latency (latency of $m + 1$) standard basis $GF(2^m)$ multiplier has been proposed in [105]. It is a semi-systolic architecture which makes use of two broadcast signals. The system level diagram of this parallel-in-parallel-out multiplier is shown in Fig. 34. This multiplier has m^2 basic cells and the structure of the basic cell is shown in Fig. 35, which has 2 2-input AND, 2 2-input XOR gates and 3 1-bit latches. The parameter $C = \sum_{k=0}^{m-1} c_k\alpha^k$, an element of $GF(2^m)$, is also an input to the multiplier so that the circuit actually performs $AB + C$.

6.2.1.2 VLSI Chip Implementation

A prototype VLSI chip was designed using CMOS 1.2 μm n-well technology. The chip layout is shown in Fig. 36. The chip implements the multiplication algorithm shown in Fig. 34 for GF(2^4). A true single phase clocking scheme [113] was used for the chip. The chip is a

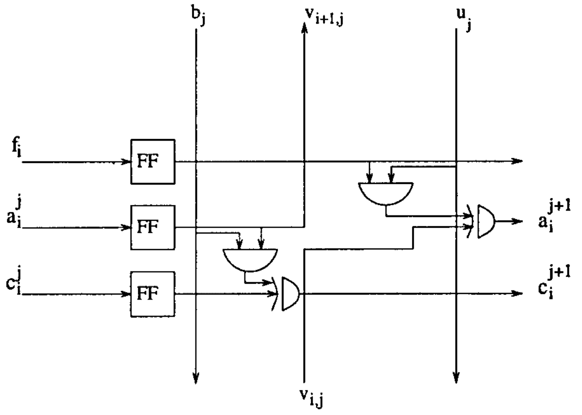


Figure 35: **Parallel-in-Parallel-out $GF(2^4)$ Multiplier**

multistage pipeline and can produce one result every clock cycle. The chip has an active area of 0.434 mm^2 and requires 1076 transistors and is programmable for different primitive irreducible polynomials. The design has been functionally verified using irsim [114]. Using Hspice simulator, the critical path was found to be 2.7ns.

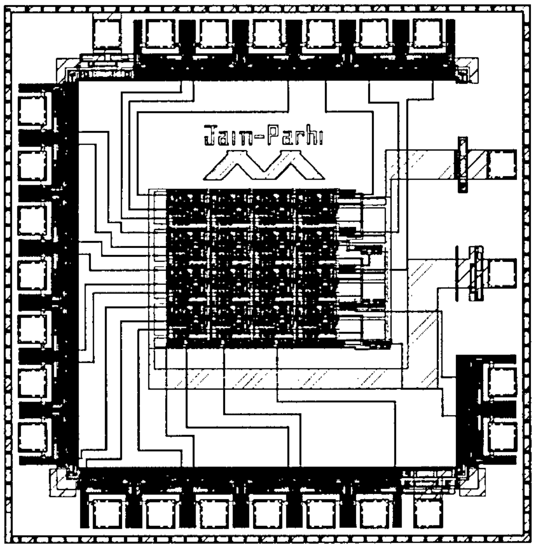


Figure 36: **Layout of the proposed multiplier chip**

6.2.1.3 Comparison with Other Multipliers

The properties of the proposed multiplier are compared in Table 19 with those of the multipliers in [115] [116]. The comparison has been done for variable multiplier and multiplicand and programmable primitive irreducible polynomials with all architectures producing 1 result

Table 19: **Comparison of Different Multipliers**

Item	**Standard Basis[4]**	**Dual Basis [17]**	**Proposed**
Number of basic cells	m^2	m	m^2
Basic Cell	2 2-input AND , 2 2-input XOR, 7 1-bit latches	2m 2-input AND, 2m 2-input XOR 3m 1-bit latches	2 2-input AND, 2 2-inputXOR 3 1-bit latches
Latency	3m	m+1	m+1
Time step	1 2-input AND and 1 2-input XOR gate	1 2-input AND and $\lceil log_2(m-1)\rceil$ 2-input XOR gate	1 2-input AND and 1 2-input XOR gate

every clock cycle. The comparison is again done in terms of 2-input gates.

It is worth noting that the proposed multiplier needs less than half the number of latches required in previous implementation [115] while maintaining the same critical path. The system latency has also been reduced to $m+1$ (assuming the outputs also are latched) from $3m$. Compared to the architecture of [116], the proposed multiplier has the same hardware and system latency but there is a reduction in the critical path from 1 AND gate followed by $\lceil log_2 m\rceil$ XOR gate to an AND gate followed by a XOR gate. The price we pay for this reduction in hardware requirement and system latency is to allow two signals to be broadcast.

6.2.2 Parallel-in-Parallel-out Squarer

6.2.2.1 Squarer Architecture

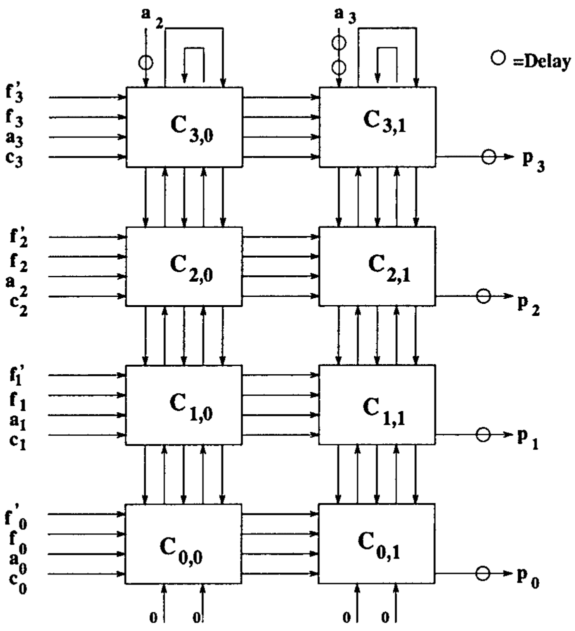
In a finite field,

$$(\alpha+\beta)^2=\alpha^2+\beta^2 \tag{53}$$

where $\alpha,\ \beta \in GF$. Using this property of finite field, we develop a hardware efficient squarer for finite field. We shall illustrate this with an example for GF(2^4). Squaring operation can be represented by

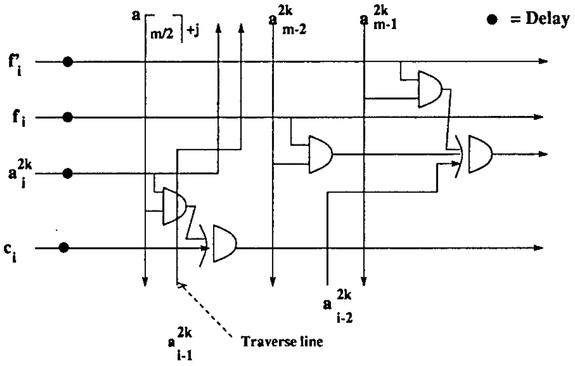
$$\begin{aligned} A &= a_0+a_1\alpha+a_2\alpha^2+a_3\alpha^3 \\ A^2 &= a_0+a_1\alpha^2+a_2\alpha^4+a_3\alpha^6. \end{aligned} \tag{54}$$

To obtain the result in the standard basis, we need to express α^4, α^6 in terms of $\{1, \alpha, \alpha^2, \alpha^3$ $\}$, i.e., in the standard basis representation. This can be achieved using the squarer shown



a2
a3
○ =Delay
f'3
f3
a3
c3
C3,0
C3,1
P3
f'2
f2
a2
c2
C2,0
C2,1
P2
f'1
f1
a1
c1
C1,0
C1,1
P1
f'0
f0
a0
c0
C0,0
C0,1
P0
0
0
0
0

Figure 37: **Parallel-in-Parallel-out Squarer**



a⌈m/2⌉+j
a2k m-2
a2k m-1
● = Delay
f'i
fi
a2k i
ci
a2k i-2
a2k i-1
Traverse line

Figure 38: **Basic Cell $C_{i,j}$ of the squarer**

in Fig. 37. The squarer consists of $m\lfloor m/2 \rfloor$ basic cells. The inputs to the squarer are $C = a_0 + a_1\alpha^2$, $B = \alpha^4$, a_2, a_3, f and f', where f' denotes the primitive polynomial f multiplied by α. The first column computes $Ba_2 + C$ and $B\alpha^2$ while the second column outputs the desired result

$$\begin{aligned} A^2 &= a_0 + a_1\alpha^2 + a_2\alpha^4 + a_3\alpha^6 \qquad (55) \\ &= C + Ba_2 + B\alpha^2 a_3. \end{aligned}$$

The basic cell in the squarer performs multiplication by α^2 and is shown in Fig. 38. The squarer is semi-systolic where each basic cell needs 4 latches. A fully systolic version would

Table 20: **Comparison of Different Approaches to Squaring**

Item	**Multiplier**	**Power-sum [117]**	**Proposed**
Number of basic cells	m^2	m^2	$m\lfloor m/2 \rfloor$
Basic Cell	2 2-input AND , 2 2-input XOR, 3 1-bit latches	3 2-input AND, 3 2-input XOR 10 1-bit latches	3 2-input AND, 3 2-inputXOR 4 1-bit latches
Latency	3m	3m	$\lfloor m/2 \rfloor + 1$
Time step	1 2-input AND and 1 2-input XOR gate	1 2-input AND and 1 3-input XOR gate	1 2-input AND and 1 3-input XOR gate

need 10 latches.

This squarer can be easily extended to a larger finite field. In general, for $GF(2^m)$, we need $\lfloor m/2 \rfloor$ columns where each column comprises of m basic cells. In the general case, the B input to the first column is α^m or α^{m+1}, i.e.,

$$\begin{aligned} B &= f,\ for\ even\ m \\ &= f',\ for\ odd\ m. \end{aligned} \tag{56}$$

Also note that we can use degenerate versions of the basic cell in the rightmost column and in the bottom row because some of the outputs are not needed.

6.2.2.2 Comparison with Other Designs

Table 20 compares the proposed squarer with a dedicated multiplier and the power-sum circuit [117]. The comparison is again done in terms of 2-input gates and all architectures produce 1 result every clock cycle.

The proposed squarer results in hardware savings of more than 50 % over using the power-sum circuit of [117] and savings of more than 25 % over a dedicated multiplier to perform the squaring operation. The system latency has been reduced to $\lfloor m/2 \rfloor + 1$ from $3m$ without any increase in the critical path.

6.2.3 Parallel-in-Parallel-out Exponentiator

6.2.3.1 Exponentiation Algorithm

Let α be an arbitrary element in GF(2^m) and we need to raise it to power $N(1 \leq N \leq 2^m-1)$. Note the range $1 \leq N \leq 2^m-1$ is sufficient to cover the entire range of N because $\alpha^{2^m-1} = 1$ and hence

$$\alpha^N = \alpha^{N \ mod \ 2^m-1}. \tag{57}$$

The exponentiation operation can be performed using the following equation:

$$\begin{aligned} \beta \ = \alpha^N \ &= \alpha^{\Sigma_{i=0}^{m-1} n_i 2^i} \\ &= \alpha^{n_0}.(\alpha^2)^{n_1}.(\alpha^{2^2})^{n_2}.\dots(\alpha^{2^m-1})^{n_{m-1}} \\ &= \prod_{i=0}^{m-1} E_i, \end{aligned} \tag{58}$$

where,

$$\begin{aligned} E_i = \ (\alpha^{2^i})^{n_i} \ &= \alpha^{2^i}, \ if \ n_i = 1 \\ &= 1, \ if \ n_i = 0 \end{aligned} \tag{59}$$

Therefore, the exponentiation operation can be performed recursively. Fig. 39 shows the flow chart of this recursive algorithm.

6.2.3.2 Exponentiator Architecture

The *square and multiply* operations in exponentiation can be implemented using the bit-level pipelined multiplier and squarers developed in the previous sections. The architecture for a bit-level pipelined exponentiator is shown in Fig. 40. This architecture consists of $(m-1)$ GF(2^m) multipliers, $(m-1)$ GF(2^m) squarers and m, m bit MUXes. The squarer SQ_i evaluates α^{2^i} for i $=1, 2, \dots, m-1$. The multiplexor MUX_i sets $E_i = \alpha^{2^i}$, if $n_i = 1$ else E_i is set to 1. The multiplier MUL_i evaluates R_i for i $= 1, 2 \dots, m-1$.

It is easy to verify that this architecture will compute $\beta = \alpha^N$. Notice again that the multiplier and squarers used are pipelined at the bit-level and hence this architecture can accept one new input every clock cycle where the clock period is determined by the delay

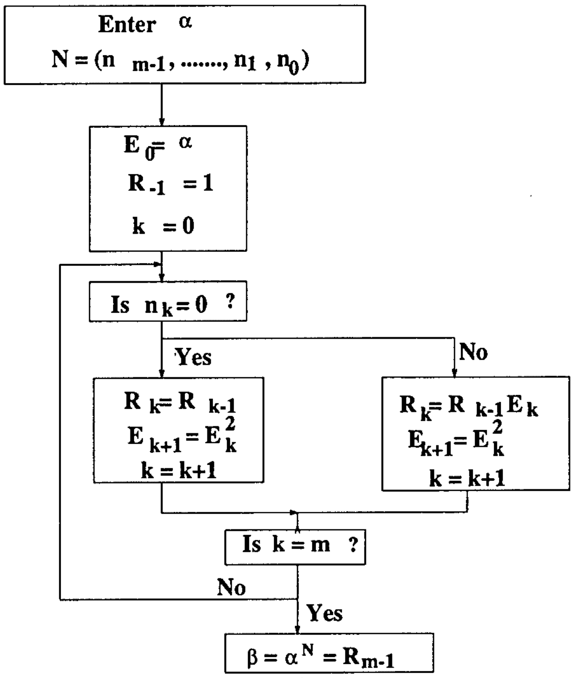

Enter α
N = (n m-1, ..., n1 , n0)
E0 = α
R-1 = 1
k = 0
Is nk = 0 ?
Yes
No
Rk = R k-1
Ek+1 = Ek^2
k = k+1
Rk = R k-1 Ek
Ek+1 = Ek^2
k = k+1
Is k = m ?
No
Yes
β = α^N = Rm-1

Figure 39: **Flow Chart for Exponentiation Operation**

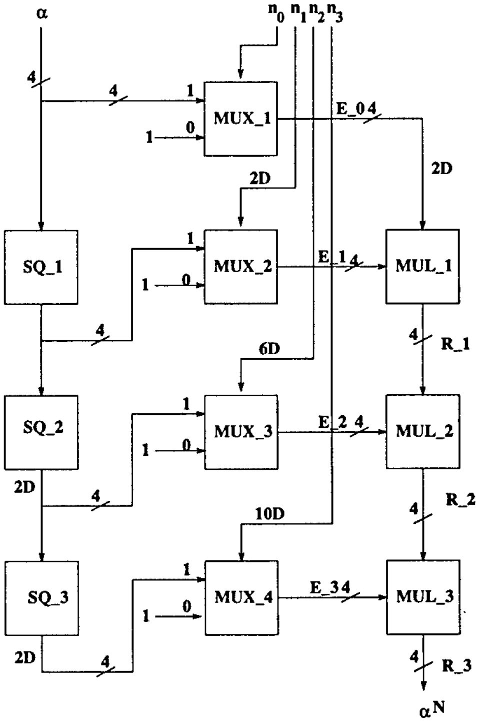

α
n0 n1 n2 n3
4
4
1
0
1
MUX_1
E_0 4
2D
2D
SQ_1
MUX_2
1
0
1
4
E_1 4
MUL_1
4
R_1
6D
SQ_2
MUX_3
1
0
1
2D
4
E_2 4
MUL_2
4
R_2
10D
SQ_3
MUX_4
1
0
1
2D
4
E_3 4
MUL_3
4
R_3
α^N

Figure 40: **Bit-level Pipelined Exponentiator**

Table 21: **Comparison of exponentiators**

Item	**Exponentiator [9]**	**Proposed**
Number of multipliers	2(m-1)	m-1
Number of cells in multiplier	m^2	m^2
Basic Cell in Multiplier	2 2-input AND , 1 3-input XOR, 7 1-bit latches	2 2-input AND, 2 2-input XOR 3 1-bit latches
Number of Squarers	-	m-1
Number of cells in squarer	-	$m\lfloor m/2 \rfloor$
Basic Cell in Squarer	-	3 2-input AND, 3 2-input XOR, 3 1-bit latches
Latency	$2m^2 + m$	$m(m-1) + \lfloor m/2 \rfloor + 1$
Time step	1 2-input AND and 1 3-input XOR gate	1 2-input AND and 1 3-input XOR gate

of a 2-input AND gate followed by a 3-input XOR gate. The architecture is a parallel-in-parallel-out architecture which can yield 1 output every clock cycle.

6.2.3.3 Architecture Comparison

The properties of the proposed bit-level pipelined exponentiator are compared with the exponentiator of [110] in Table 21. Both the architectures produce 1 result every clock cycle.

It is worth noting that in the proposed exponentiator architecture, the primitive irreducible polynomial f can be shared between the multiplier and the squarer. This effectively reduces the number of latches needed in each basic cell of the squarer to 3.

From this table, we can see that the proposed exponentiator results in hardware savings of 12.5% over [110]. We have also reduced the system latency to $m(m-1) + \lfloor m/2 \rfloor + 1$ from $2m^2 + m$ without any change in the critical path.

6.3 Efficient Standard Basis Reed-Solomon Encoder

An efficient Reed-Solomon (RS) recoder has also been presented during this project [106]. The hardware complexity is identical to the well-known Berlekamp dual basis encoder. However, it offers two advantages -a critical path independent of the order of RS codes being

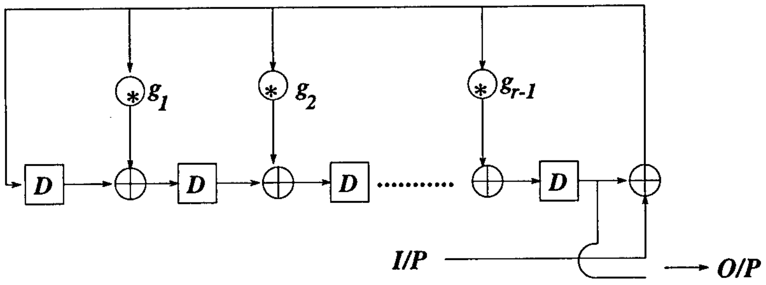


Figure 41: **System Diagram of Reed-Solomon Encoder**

implemented and the ability to encode without any need for basis conversion.

6.3.1 Reed-Solomon Encoding Algorithm

Systematic RS encoding can be described by

$$v(x) = x^{n-k}u(x) + < x^{n-k}u(x) >_{g_t(x)}, \tag{60}$$

where $g_t(x)$ is the generator polynomial of a t-error correcting RS code, $< x^{n-k}u(x) >_{g_t(x)}$ denotes the remainder when $x^{n-k}u(x)$ is divided by $g_t(x)$. This equation assures that $v(x)$ is a multiple of $g_t(x)$ and the code is systematic. The block diagram of RS encoder is given in Fig. 41.

6.3.2 Reed-Solomon Encoder

The proposed RS encoder is based on the new semi-systolic multiplier in [105]. This standard basis multiplier computes $A, A\alpha, \cdots, A\alpha^{m-1}$ in sequence and performs a scalar multiplication of these vectors with $b_0, b_1, \cdots, b_{m-1}$, respectively. These partial products are added to compute the multiplication AB. The computation of the vectors $A, A\alpha, \cdots, A\alpha^{m-1}$ can be shared if we need to multiply one term A with a number of terms simultaneously. Suppose, we need to compute $P_1 = AB_1$ and $P_2 = AB_2$ at the same time. Then,

$$\begin{aligned} P_1 &= AB_1 = \sum_{k=0}^{m-1}(A\alpha^k)b_{1k} \\ P_2 &= AB_2 = \sum_{k=0}^{m-1}(A\alpha^k)b_{2k}. \end{aligned} \tag{61}$$

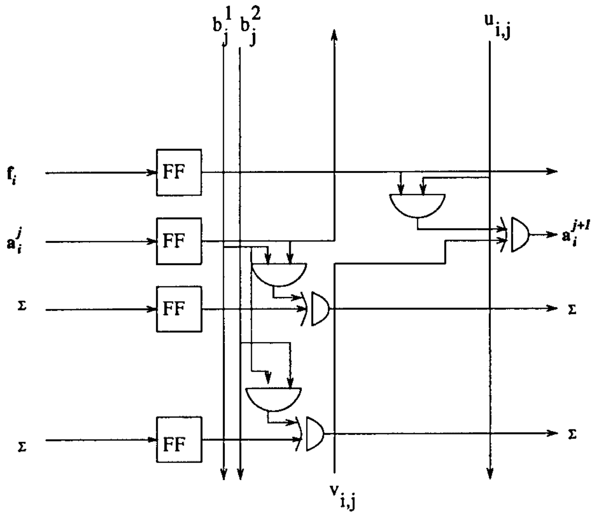


Figure 42: **Basic Cell $C_{i,j}$ of the Proposed Reed-Solomon Encoder**

The above equation illustrates that the computation of $A, A\alpha, \cdots, A\alpha^{m-1}$ can be shared between the computation of the products P_1 and P_2. Once these vectors are computed, they can be simultaneously multiplied to B_1 and B_2 to obtain the products P_1 and P_2. This approach can also be extended to any number of simultaneously multiplications. As can be seen from Fig. 41, there exist a broadcast signal in RS encoder, which is composed of m bits and this symbol is multiplied by r coefficients of the generator polynomial $g_t(x)$ simultaneously. Therefore, the substructure sharing idea can be utilized to derive efficient RS encoder.

The basic cell for implementing (61) is shown in Fig. 42, which can also be used as the basic cell for RS encoder. For RS encoder over $GF(2^4)$, the proposed structure consists of 16 cells identical to the basic cell shown in Fig. 42. The critical path consists of 1 2-input AND gate followed by 1 2-input XOR gate.

In general, for a (n, k) RS encoder with symbols from $GF(2^m)$, the proposed encoder would consist of m^2 basic cells. Each basic cell would have $(n-k+1)$ 2-input AND gates, $(n-k+1)$ 2-input XOR gates and (n-k+2) 1-bit latches. The critical path in the proposed RS encoder is independent of the order of the RS code being implemented and is equal to the propagation delay of 1 2-input AND gate followed by a 2-input XOR gate.

Table 22: **Comparison of exponentiators**

Item	**Berlekamp's**	**Proposed**
Number of basic cells	m	m^2
Basic Cell	$m(r+1)$ 2-input AND, $m(r+1)$ 2-input XOR, $m(r+2)$ 1-bit latches	$r+1$ 2-input AND, $r+1$ 2-input XOR $r+2$ 1-bit latches
Latency	$m+1$	$m+1$
Critical Path	1 2-input AND and $\lceil log_2(m-1) \rceil$ 2-input XOR gates	1 2-input AND and 1 2-input XOR gate
Basis Conversion	Yes	No

6.3.3 Comparison with Berlekamp's Dual Basis RS Encoder

The properties of the proposed RS encoder are compared with the well-known Berlekamp's encoder in Table 22. The comparison is done for a (n, k) RS encoder over $GF(2^m)$ and the generator polynomial having $r = n - k$ coefficients. The generator polynomial and the primitive polynomial are both programmable.

6.4 Efficient Finite Field Serial/Parallel Multiplication

6.4.1 Bit-Serial Finite Field Multiplier

A new bit-serial/parallel finite field multiplier has been presented in [107] with standard basis representation. This design is regular and well suited for VLSI implementation. As compared to existing serial/parallel finite field multipliers, it has smaller critical path, lower latency and can be easily pipelined. When it is used as a building block for large systems, it can achieve more savings in hardware in the broadcast structures by utilizing sub-structure sharing techniques which has been introduced in last section [106].

6.4.1.1 Multipier Architecture

The proposed design is semi-systolic with bi-directional data flow. It utilizes LSB first implementation based on the following equation

$$\begin{aligned} C &= AB \; mod \; f(x) \\ &= (Ab_0 \; mod \; f(x)) + (Ab_1\alpha \; mod \; f(x)) \end{aligned}$$

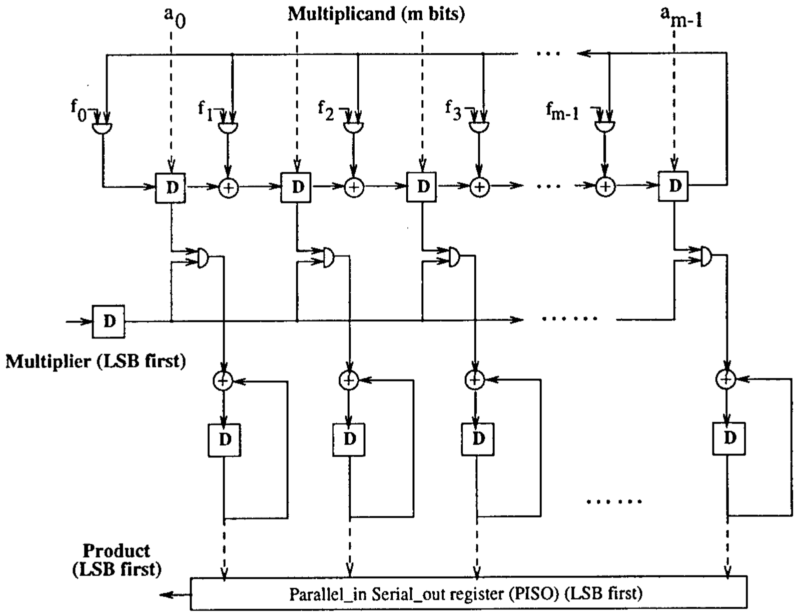

a0
Multiplicand (m bits)
am-1
f0
f1
f2
f3
fm-1
D
D
D
D
D
Multiplier (LSB first)
D
D
D
D
Product
(LSB first)
Parallel_in Serial_out register (PISO) (LSB first)

Figure 43: Bit Serial/Parallel Multiplication Circuit

$$\begin{aligned} & +(Ab_2\alpha^2 \; mod \; f(x)) + \cdots \\ & +(Ab_{m-1}\alpha^{m-1} \; mod \; f(x)) \\ = \; & b_0A + b_1(A\alpha \; mod \; f(x)) \\ & +b_2(A\alpha^2 \; mod \; f(x)) + \cdots \\ & +b_{m-1}(A\alpha^{m-1} \; mod \; f(x)), \end{aligned} \tag{62}$$

or

$$\begin{aligned} (A\alpha^{k-1})\alpha &= A\alpha^k \\ A\alpha^{k-1}b_{k-1} \; + \; C^{(k-1)} &= C^{(k)}, \end{aligned} \tag{63}$$

where $C^{(k)} = \sum_{i=0}^{k-1} Ab_i\alpha^i$, and $C^{(0)}$=0. Fig. 43 shows the overall architecture.

It contains three parts. The upper part is a linear feedback shift register (LFSR) with f_i's as the coefficients. The concepts and properties of LFSR can be found in [118]. Here it is used to perform one-bit shifting followed by *mod* $f(x)$ operation, which essentially is *multiplication by* α. The middle part is partial product generator. The lower part is accumulation part.

Bits of multiplicand are loaded to LFSR in parallel every m clock cycles. Multiplier bits are loaded serially. Once one multiplication is complete, the final product is transfered to

Table 23: Comparison with Systolic-array based Architecture

Properties	**Wang et al. [119]**	**Proposed**
Resources	3m 2-input AND m 3-input XOR 9m latches m 2-to-1 MUXes	2m 2-input AND (2m-1) 2-input XOR (4m+2) latches 4m 2-to-1 MUXes
# of Transistors	176m	104m+24
Latency	3m	m+1
Critical Path	1 2-input AND + 1 3-input XOR	1 2-input AND + 1 2-input XOR

a parallel-in serial-out (PISO) register and shifted out serially. One control signal is needed for I/O multiplexing and initializing accumulation latches once every m clock cycles.

6.4.1.2 Comparison with Other Designs

The comparison in this section is based on the following assumption.

- Multiplication is over $GF(2^m)$ and primitive polynomial $f(x)$ is programmable.
- Both multiplicand and multiplier are assumed to be programmable for flexibility considerations.
- 3-input XOR gate is implemented using two 2-input XOR gates.

The properties of the proposed serial/parallel multiplier are compared with one systolic array realization [119] in Table 23. The proposed design has less number of latches and has a smaller latency.

This design is also compared with existing serial/parallel architectures in Table 24. All multipliers in Table 24 make use of broadcast signals. The I/O cost, including serial/parallel converter and MUXes are ignored in Table 24 because all these three designs have the same I/O cost. It is worth noting that the proposed design has smaller critical path and smaller latency compared with [120]. The multiplier in [121] is based on MSB first algorithm, i.e., the multiplier bits are loaded serially with most significant bit first. The disadvantage is that it performs multiplication by α and accumulation in serial every clock cycle, hence need a 3-input Xor gate in accumulation part. Furthermore, when used as a building block for a

Table 24: Comparison with other serial/parallel Architectures

Properties	[121]	**Hasan et al. [120]**	**Proposed**
Resources	2m MUXes (m-1) 3-input XOR 1 2-input XOR (4m+2) latches	(3m-1) 2-input AND (3m-2) 2-input XOR (4m+1) latches 1 switch	2m 2-input AND (2m-1) 2-input XOR (4m+2) latches
# of transistors	88m+24	100m-2	88m+24
Latency	m+1	2m+2	m+1
Critical Path	1 2-input AND 1 3-input XOR	1 2-input AND + log_2(m-1) 2-input XOR	1 2-input AND + 1 2-input XOR
Hardware Utilization	100%	50%	100%

larger system, the proposed multiplier can achieve less than linear increase in hardware as the number of multipliers increases by substructure sharing. However, there is no straightforward way to apply substructure sharing technique to the multiplier in [121].

6.4.2 Generalized Serial/Parallel Finite Field Multiplication

In [107], two general digit-serial multiplication algorithms are presented. They can be used to derive efficient bit-parallel algorithms for finite field serial/parallel multiplication. The optimal primitive polynomials over $GF(2^m)$ (for $2 \leq m \leq 9$) are provided which will generate structures with minimum hardware complexity and relatively more flexibilities for feasible digit-sizes. A multiplier over $GF(2^m)$ has been given as an example in [107] showing how to derive efficient multiplier structures using the proposed algorithm. This multiplier has less number of transistors, smaller critical path and consumes less power compared to the existing semi-systolic architecture.

6.4.2.1 Digit-Serial Multiplication Algorithms

Assume digit-size $= D$. Let d denote the total number of digits and $d = \lceil m/D \rceil$. Let $A = \sum_{i=0}^{m-1} a_i\alpha^i, B = \sum_{i=0}^{d-1} B_i\alpha^{Di}$, where

$$B_i = \begin{cases} \sum_{j=0}^{D-1} b_{Di+j}\alpha^j & , \quad for\ 0 \leq i \leq d-2 \\ \sum_{j=0}^{m-1-D(d-1)} b_{Di+j}\alpha^j & , \quad for\ i = d-1 \end{cases} .$$

Then $C = A \cdot B \ mod \ f(x) \ = A \cdot \sum_{i=0}^{d-1} B_i \alpha^{Di} \ mod \ f(x)$. We have the following two equations:

$$\begin{aligned} C &= (\ AB_0 + A\alpha^D B_1 + A\alpha^D \cdot \alpha^D B_2 + \cdots \\ &\quad + A\alpha^{D(d-2)} \cdot \alpha^D B_{d-1} \ ) \ mod \ f(x)) \\ &= (\ B_0 A + B_1 (A \cdot \alpha^D \ mod \ f(x)) \\ &\quad + B_2 (A\alpha^D \cdot \alpha^D \ mod \ f(x)) + \cdots \\ &\quad + B_{d-1}(A\alpha^{D(d-2)} \cdot \alpha^D \ mod \ f(x) \ ) \ mod \ f(x) \end{aligned} \tag{64}$$

for the least significant digit (LSD) first scheme and

$$\begin{aligned} C &= (\, (\, (\, (\, (\cdots (\, (\, (AB_{d-1} \ mod \ f(x) \ ) \cdot \alpha^D + AB_{d-2} \ ) \ mod \ f(x) \ ) \cdot \alpha^D \\ &\quad + \cdots \ ) \cdot \alpha^D) + AB_1) \ mod \ f(x) \ )\alpha^D + AB_0 \ ) \ mod \ f(x) \end{aligned} \tag{65}$$

for the most significant digit (MSD) first scheme. Hence we have two algorithms for digit-serial/parallel multiplication.

Algorithm 1 (LSD first)

1. $C^{(0)} \ = \ 0$, for $i = 0$;

2. At ith iteration, $(1 \leq i \leq d-1)$

$$\begin{aligned} (A\alpha^{D(i-1)}) \cdot \alpha^D \ mod \ f(x) \ &= \ A\alpha^{Di} \ mod \ f(x) \\ (A\alpha^{D(i-1)} B_{i-1}) \ + \ C^{(i-1)} \ &= \ C^{(i)}, \end{aligned} \tag{66}$$

where $C^{(i)} = \sum_{j=0}^{m+D-2} C_j^{(i)} \alpha^j$, $A\alpha^{Di} \ mod \ f(x) = \sum_{j=0}^{m-1} a_j^{(i)} \alpha^j$;

3. At dth iteration,

$$A\alpha^{D(d-1)} \cdot B_{d-1} + C^{(d-1)} = C^{(d)}; \tag{67}$$

4. Correction. Product of A and B is $(C^{(d)} \ mod \ f(x))$.

□

Algorithm 2 (MSD first)

1. $C^{(0)} \;=\; 0$, for $i = 0$;

2. At ith iteration, $(1 \leq i \leq d)$

$$C^{(i)} \;=\; (\; C^{(i-1)} \cdot \alpha^{D} + AB_{d-i} \;)\; mod\; f(x), \tag{68}$$

where $C^{(i)} = \sum_{j=0}^{m-1} C_j^{(i)} \alpha^j$.

□

Two essential steps in above algorithms are computing the partial product $A \cdot B_i$, and computing the *mod* $f(x)$ reduction. Computation of $A \cdot B_i$ can be performed using direct Boolean AND and XOR operations at bit level. However, the computation of *mod* $f(x)$ operation is highly dependent on the primitive polynomial $f(x)$ and is much more involved. An algorithm for simplifying this *mod* $f(x)$ operation is provided.

Theorem 1. Assume $f(x) = x^m + x^k + \sum_{i=0}^{k-1} f_i x^i$. For $t \leq m - 1 - k$, the coordinates of α^{m+t} can be obtained by the following equation,

$$\begin{aligned} \alpha^{m+t}\; mod\; f(x) \;&= (\; \alpha^m\; mod\; f(x)\;) \cdot \alpha^t \\ &= (\; \alpha^k \;+\; \sum_{i=0}^{k-1} f_i \alpha^i\;) \cdot \alpha^t. \end{aligned} \tag{69}$$

Theorem 2. For digit-size $D \leq m - k$, the *mod* $f(x)$ reduction operation for digit-serial multiplication with digit-size D can be performed as follows:

$$\alpha^{m+t}\; mod\; f(x) \;=\; (\alpha^k + \sum_{i=0}^{k-1} f_i \alpha^i) \cdot \alpha^t, for\; t \leq D - 1. \tag{70}$$

Therefore, according to Theorem 2, when $D \leq m - k$, the *mod* $f(x)$ operation in step 2 and 4 of Algorithm 1 and step 2 of algorithm 2 can be accomplished by simply taking the higher order digits (HD, from bit m to bit $m + D - 1$) of the partial product, multiplying it by $(\alpha^k + \sum_{i=0}^{k-1} f_i \alpha^i)$ and adding it to lower order digits (LD, from bit 0 to bit $m - 1$) of the partial product. Then the highest degree of the result will be guaranteed to be less than m.

It needs to be pointed out that for some finite field $GF(2^m)$ over which primitive polynomial of the form $x^m + x + 1$ exists, the digit-size D can vary from 1 to m instead of from 1 to $m - 1$.

Table 25: List of Optimal polynomials over $GF(2^m)$

finite field	*prim. polynomial*	*feasible digit-size*
m=2	$f(x) = x^2 + x + 1$	$1 \leq D \leq 2$
m=3	$f(x) = x^3 + x + 1$	$1 \leq D \leq 3$
m=4	$f(x) = x^4 + x + 1$	$1 \leq D \leq 4$
m=5	$f(x) = x^5 + x^2 + 1$	$1 \leq D \leq 3$
m=6	$f(x) = x^6 + x + 1$	$1 \leq D \leq 6$
m=7	$f(x) = x^7 + x + 1$	$1 \leq D \leq 7$
m=8	$f(x) = x^8 + x^4 + x^3 + x^2 + 1$	$1 \leq D \leq 4$
m=9	$f(x) = x^9 + x + 1$	$1 \leq D \leq 9$

The optimal primitive polynomials over $GF(2^m)$, for $2 \leq m \leq 9$, are given in Table 25. Optimal primitive polynomials are chosen keeping in mind the simplicity of architectures and flexibilities on feasible digit-sizes when the proposed algorithms are used.

6.4.2.2 Multiplier over $GF(2^8)$

Let $f(x) = x^8 + x^4 + x^3 + x^2 + 1$ be the primitive polynomial over $GF(2^8)$ and α be the root of $f(x)$. A multiplier has been derived using the proposed LSD first algorithm. The overall structure as well as basic cells of $GF(2^8)$ multiplier are shown in Fig. 44. Here the multiplier is in digit-parallel form, from which the corresponding digit-serial architecture can be easily derived.

It is worth noting that more substructure sharing can be achieved when the *mod* $f(x)$ operation is performed in the proposed way, which is illustrated by the shaded regions in Fig. 44.

This multiplier has been compared with the existing semi-systolic architecture [105] under the assumption that both multipliers are over $GF(2^8)$ with primitive polynomial $f(x) = x^8 + x^4 + x^3 + x^2 + 1$. A hierarchical energy analysis tool HEAT [122] was used to compute the average power for the two multipliers (with and without pipelining). It was found that the best case for semi-systolic multiplier over $GF(2^8)$ was the one with 4-bit level pipelining, the best case for proposed multiplier was the one without pipelining. Therefore, the comparison was made between the two multipliers for both non-pipelining and 4-bit pipelining cases. All results are summarized in Table 26.

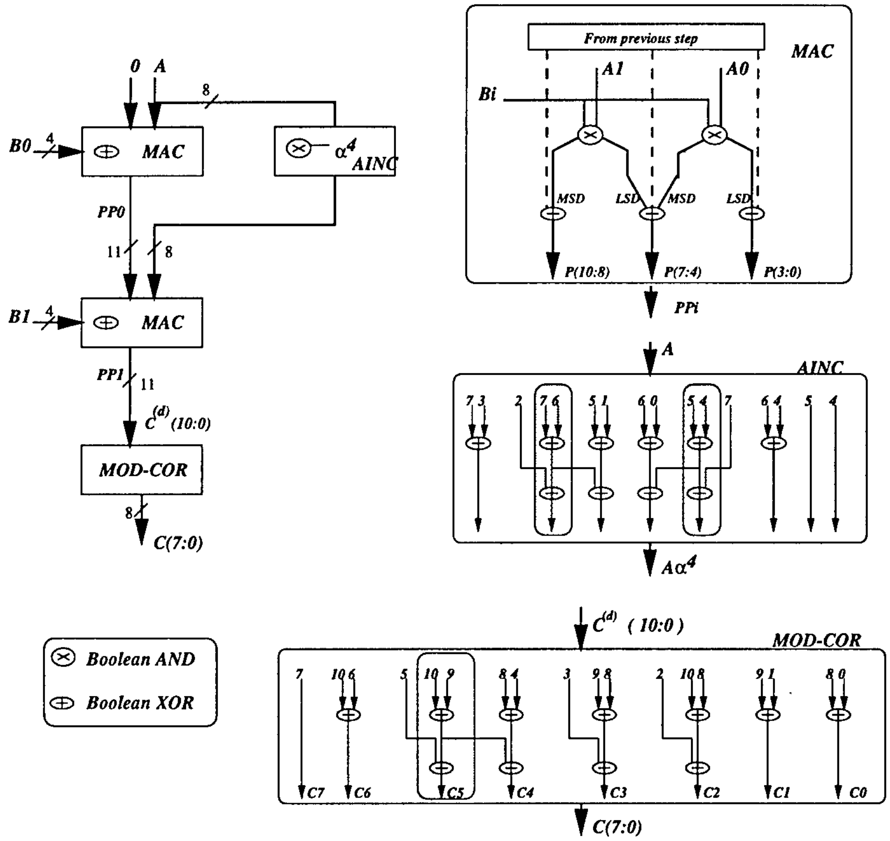

0
A
8
B0
4
MAC
α^4
AINC
PP0
11
8
B1
4
MAC
PP1
11
C^(d) (10:0)
MOD-COR
8
C(7:0)
Boolean AND
Boolean XOR
From previous step
MAC
Bi
A1
A0
MSD
LSD
MSD
LSD
P(10:8)
P(7:4)
P(3:0)
PPi
A
AINC
Aα^4
C^(d) (10:0)
MOD-COR
C7
C6
C5
C4
C3
C2
C1
C0
C(7:0)

Figure 44: digit-Serial/Parallel Multiplication Circuit over $GF(2^8)$

Table 26: Comparison between proposed multiplier and semi-systolic multiplier for m=8

	Semi-systolic [105]		**Proposed**	
Properties	4-bit pipe.	non pipe.	4-bit pipe.	non pipe.
Resources	64 2-input AND 80 2-input XOR 24 latches	64 2-input AND 80 2-input XOR 8 latches	64 2-input AND 73 2-input XOR 38 latches	64 2-input AND 73 2-input XOR 8 latches
# trans.	1280	1024	1448	968
Latency	2 clk cycles	1 clk cycles	3 clk cycles	1 clk cycles
Critical Path	1 2-input AND 4 2-input XOR	1 2-input AND 8 2-input XOR	1 2-input AND 3 2-input XOR	1 2-input AND 7 2-input XOR
Power Consump. (μW)	889	1198	708.18	578.56

From Table 26 we can conclude that the proposed multiplier without pipelining gives the best overall performance.

7 Order-Configurable, Power Efficient FIR Filters

With the recent explosion of portable and wireless real-time, digital signal processing applications, the demand for low-power circuits has increased tremendously [123]-[125]. This demand has been satisfied by utilizing ASICs; however, ASICs allow for very little reconfigurability. Another new trend is the need to minimize the design cycle time. Therefore many programmable logic devices (PLDs) (e.g., field-programmable gate arrays) are being utilized for prototyping and even production designs [126]. The main disadvantage of these PLDs is that they suffer from slow performance because their architectures have been optimized for random logic and not for digital signal processing implementations. In this paper, a solution for the implementation of high-speed, low-power, and order-configurable finite impulse response (FIR) filters is presented. This architecture was designed by applying the folding and the retiming transformations and the filter order can vary from 1 to 31 using one chip. Multiple chips can be cascaded to achieve higher order FIR filters.

This new architecture consists of two parts: a configurable processor array (CPA) [127] and a phase locked loop (PLL). The CPA contains the multiply-add functional units and the PLL is designed to automatically vary the internal voltage to match the desired throughput rate and minimize the peak power dissipated by the CPA. We utilize a novel programmable divider and a voltage level shifter in conjunction with the clock to control the internal supply voltage. The CPA portion contains folded multiply-add (FMA) units which operate in two phases: the configuration phase where the processor array is programmed for a specific sample-rate and filter-order, and the execution phase where the processor array performs the desired filtering operation. We also implemented novel programmable subcircuits that provides the order configurability of the architecture. This design has been implemented using Mentor Graphics tools and 1.2μm CMOS technology.

In section 7.1, we briefly describe how the CPA is derived and the design parameters. In section 7.2, the design of the CPA components are described in more detail and section 7.3 describes the PLL components. Simulation results are provided in section 7.4 to demonstrate the effectiveness of the design and the power savings.

7.1 Background

Consider the transpose-form architecture of a 6-tap FIR filter that realizes the function $y(n) = a_0x(n) + a_1x(n-1) + a_2x(n-2) + a_3x(n-3) + a_4x(n-4) + a_5x(n-5)$. If we implement this 6-tap filter using 2 multiply-add functional units, which corresponds to using a folding factor of 3 [128], (i.e., 3 multiply-add operations are folded to the same functional unit), we will have a folded architecture shown in Fig. 45. This architecture consists of folded multiply-add units (FMA). The inputs and outputs ($x(n)$ and $y(n)$) to each FMA will hold the same sample data for three clock cycles before changing to the next sample. To completely pipeline the folded architecture, additional delays are introduced at the input ($x(n)$) by using the retiming transformation [129] along with pipelining. This

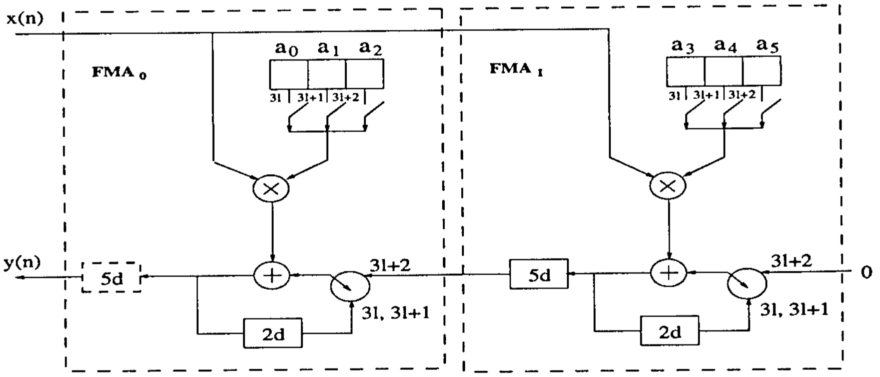


Figure 45: **The folded architecture of the 6-tap FIR filter (folding factor = 3).**

modified structure is now periodic with a period of three clock cycles (or 3-periodic). This technique can be applied to any N-tap FIR filter for any folding factor, p.

To achieve programmability and the CPA architecture, we convert the fixed number of registers in Fig. 45 into programmable delays that are constrained by a maximum folding factor p_{max} as shown in Fig. 46. To implement an N-tap filter using this architecture, a total of M (where $M = \lceil N/p \rceil$) FMA modules are required. This CPA architecture is a periodic system with period p; therefore it is designed to produce filter outputs from module FMA_0 in clock cycles ($t \bmod p) = 0$ (where t = time in clock cycles) and hold it for p cycles. Note that mux4 in Fig. 46 is only required for module FMA_0 to hold the filter output data for p clock cycles and is redundant in the other FMA_j modules ($j \neq 0$). These other multiplexers can be replaced by a single delay along with sharing of the $(p-1)$ registers in the feedback accumulation path. The switching times of all of the programmable multiplexers are summarized in Table 27.

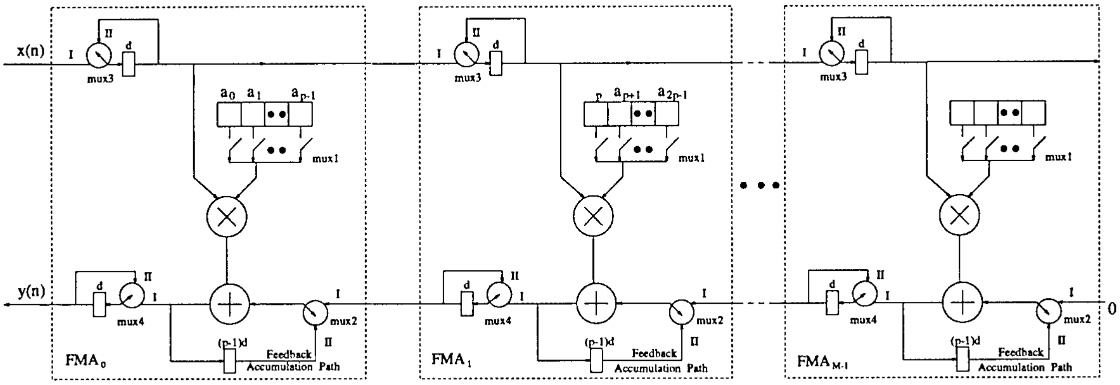


Figure 46: **A configurable processor array (CPA) for N-tap FIR filters which is p-periodic.**

mux#	mux definition
1	a_i in clock cycle $((p-1)(j+1)+i) \bmod p$
2	I in clock cycle $((p-1)(j+1)-1) \bmod p$
3	I in clock cycle $((p-1)(j+1)-1) \bmod p$
4	I in clock cycle $((p-1)(j+1)) \bmod p$

Table 27: Multiplexer definitions

Before implementing this general structure, we had to set values for N_{max} and p_{max}. We chose to set N_{max} (maximum number of taps) to 32 because an FIR filter will provide good performance for filter lengths around 32. We set p_{max} (maximum folding factor) to 8 because we wanted p_{max} to be a power of 2 and desired greater flexibility with minimal control overhead. With $N_{max} = 32$ and $p_{max} = 8$, a total of 4 FMA modules needed to be integrated onto a single chip.

7.2 Configurable Processor Array

The 8-bit parallel multiplier is a key part of the CPA module because it determines the critical path of the system. We chose to utilize the Baugh-Wooley algorithm for the multiplier because the control overhead is smaller than other algorithms (e.g., Booth recoding) and the full-adders are not wasted on sign extensions. This algorithm generates a matrix of partial product bits and a fast multi-operand adder [130] was employed to accumulate these partial products. To minimize the critical path in the accumulation path, we used the Wallace tree approach [131]. In the CPA design of Fig. 46, we see that the feedback accumulation path requires $p-1$ synchronization registers. Because p is a programmable parameter, $p-1$ can range from 0 to 7 ($p_{max}-1$), we implemented them as a programmable delay line as shown in Fig. 47. Each delay line contains seven 8-bit registers, seven 8-bit multiplexers, and one control unit. The control unit is a simple decoder, that converts p into seven control bits and each control bit directs the data through or around a delay.

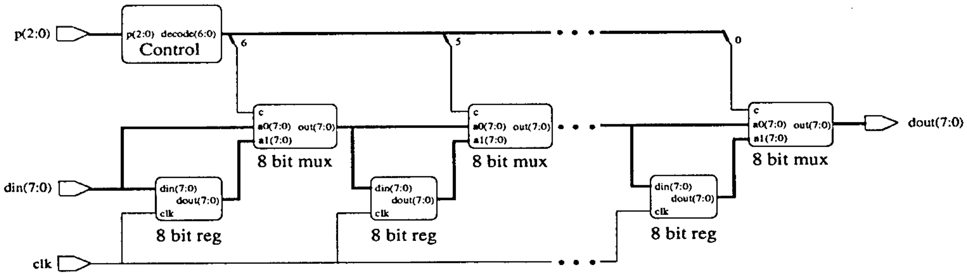


Figure 47: $p-1$ **programmable delay line.**

The multiplexers mux2, mux3 and mux4 shown in Fig. 46 are 2-to-1 p-periodic multiplexers. Their functions are to select input I in one of p clock-cycles. These multiplexers use a 3-bit ($\lceil \log_2(p_{max}) \rceil$-bit) binary counter with asynchronous reset and synchronous parallel load. In addition, two 3-bit registers and a comparator are used in the control circuitry of each multiplexer. One register holds p and the second holds a programmed clock cycle value ranging from 0 to $p-1$. When the counter output equals the held clock cycle value, the controller allows the data on I to pass to the output. The final multiplexer in Fig. 46, mux1, is a programmable p-to-1 p-periodic multiplexer which consists of one 8-bit 8-to-1 multiplexer and one control unit. At each counter state one of p control lines will be high to activate the p-to-1 multiplexer.

7.3 Phase Locked Loop

Reducing the supply voltage of VLSI chips is commonly used to save power; however, it also slows down the critical path of the circuit. If the supply voltage is reduced too much, the critical path will become too slow to assure correct functionality of the design. Therefore we designed a phase locked loop (PLL) circuit that automatically controls the internal supply voltage to provide the lowest voltage allowable while still achieving the throughput required for the application [132]. The PLL consists of a phase detector, a charge pump with a loop filter, a voltage controlled oscillator (VCO), a programmable divider, and a voltage level shifter. All of these components form a feedback circuit that automatically adjusts the voltage level as required by the programmed parameters and the clock speed.

The schematic of the programmable divider used in the PLL is shown in Fig. 48. To achieve a 50% duty cycle, we had to accommodate three possible cases of p. If p is 1, the input clock simply passes through the divider without any change. For even p, the divider toggles its output every $p/2$ input clock cycles by using a programmable counter. When p is odd ($p > 1$), the divider must alter the output every $(p-1)/2 + 1/2$ input clock cycles. This means the output may toggle at the rising edge and falling edge of the input clock. To detect the edge where the divider should toggle its output, we utilize two programmable counters; one to detect rising edges, and the other to detect falling edges. These counters generate a series of pulses representing edges and an OR gate combines them into a single pulse. Finally the Toggle component alters the output according to the pulses generated by the OR gate. The two multiplexers in Fig. 48 select the appropriate clock output from the three cases depending on the value of p.

The function of the voltage level shifter (VLS) is to raise the output voltage of the loop filter to a usable level in the CPA. By sizing transistors in the VLS, we can adjust the

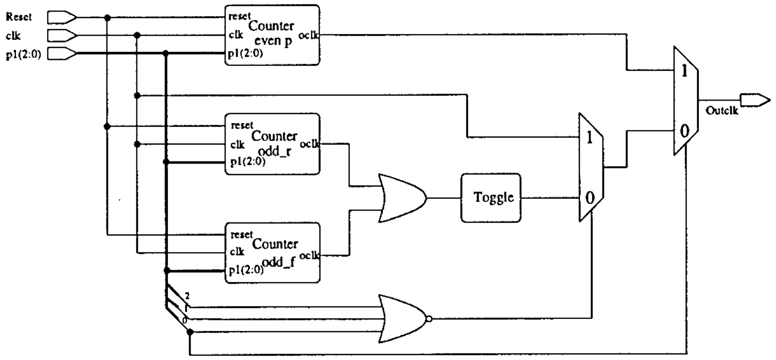


Figure 48: **programmable divider**

amount of voltage that will be shifted (known as the voltage shift level). However, the power consumption of the voltage level shifter will increase with an increase in the the voltage shift level. So there is a tradeoff between power consumption and the voltage shift level. Our experiments have shown that a shift of 0.6V provided enough internal voltage to safely operate the CPA within the design specifications while minimizing the power consumption.

7.4 Simulation

Using Mentor Graphics tools, simulations determined the critical path of the design to be 7ns at the schematic level which means that it is safe to operate the architecture up to 100 MHz. The CPA was designed to be operated with sample rates in the range of 10MHz to 100MHz, which corresponds to an internal clock rate of 1.125MHz (with $p = 8$) to 100MHz (with $p = 1$). This range of frequencies corresponds to an internal power supply range of 4.5V to 2.0V. Efficient power consumption is one of the important features of our design and Table 28 shows the power consumptions in mW for each CPA component at different frequencies and power supplies. From Table 28, we can see that at 100MHz, the power consumption of the CPA without the PLL and using a 5V supply voltage will consume 1101.48mW. By utilizing the PLL supply voltage for 100MHz (4.5V) the power consumption can be reduced to 863.32mW. At 10MHz, we can save 95.37mW by using the PLL supply voltage automatically generated for 10MHz verses a 5V supply. Of course the PLL will consume some power of its own and results of power consumption simulations for the various components of the PLL are listed in Table 29. From Table 29, we can see that even if we include the power consumption of the PLL, we will still save 210.06mW at 100MHz, and

Component	5V, 100MHz	4.5V, 100MHz	5V, 10MHz	2.0V, 10MHz
multiplier	140.6	112.5	14.23	1.98
pmux(p-1)	5.17	3.85	1.14	0.050
adder	18.8	16.52	2.18	0.28
p1delay	60	43.2	6.03	0.77
pmux(2-1)	11.6	9.5	0.65	0.063
1delay	8	5.63	0.9	0.099
FIR(digital)	1101.48	863.32	109.24	13.87

Table 28: Power consumption for digital parts of FIR filter in mW

81.79mW at 10MHz.

Component	phase detector	charge pump loop filter	VCO	level shifter	divider	total
100MHz	8.3	7.55	14.875	0.9	1.345	28.1
10MHz	2.68	0.355	3.335	0.999	1.34	13.58

Table 29: Power consumption for PLL parts in mW

8 List of Publications Supported by RASSP

- C.Y. Wang, and K.K. Parhi, "The MARS High-Level DSP Synthesis System", in *VLSI Design Methodologies for Digital Signal Processing Architectures*, edited by M. Bayoumi, pp. 169-205, Kluwer Academic Press, 1994
- S. Jain and K.K. Parhi, "Efficient Power Based Galois Field Arithmetic Architectures", in *VLSI Signal Processing VII*, pp. 306-315, IEEE Press, Oct. 1994 (Proc. of the Seventh IEEE VLSI Signal Processing Workshop, La Jolla, CA)
- K.K. Parhi, "High-Level Transformations for DSP Synthesis", Chapter 8.1 in *Microsystems Technology for Multimedia Applications: An Introduction*, edited by B. Sheu *et al.*, IEEE ISCAS-95 Tutorial Book, pp. 575-587, IEEE Press, 1995
- C.-Y. Wang and K.K. Parhi, "High-Level DSP Synthesis", Chapter 8.4 in *Microsystems Technology for Multimedia Applications: An Introduction*, edited by B. Sheu *et al.*,

IEEE ISCAS-95 Tutorial Book, pp. 615-627, IEEE Press, 1995

- T.C. Denk and K.K. Parhi,"Systematic Design of Architectures for M-ary Tree-Structured Filter Banks", pp. 157-166, in *VLSI Signal Processing VIII*, IEEE Press, October 1995 (Proc. of the 1995 IEEE Workshop on VLSI Signal Processing, Sakai, Japan)
- K. Ito and K.K. Parhi,"Register Minimization in Cost-Optimal Synthesis of DSP Architectures", pp. 207-216, in *VLSI Signal Processing VIII*, IEEE Press, October 1995 (Proc. of the 1995 IEEE Workshop on VLSI Signal Processing, Sakai, Japan)
- C.-Y. Wang, and K.K. Parhi, "High-Level DSP Synthesis using Concurrent Transformations, Scheduling, and Allocation", *IEEE Transactions on Computer Aided Design*, **14**(3), pp. 274-295, March 1995
- C.-Y. Wang, and K.K. Parhi, "Resource Constrained Loop List Scheduler for DSP Algorithms", *Journal of VLSI Signal Processing*, **11**(1/2), pp. 75-96, October 1995
- K. Ito and K.K. Parhi, "Determining the Minimum Iteration Period of an Algorithm", *Journal of VLSI Signal Processing*, **11**(3), pp. 229-244, December 1995
- T.C. Denk and K.K. Parhi, "Lower Bounds on Memory Requirements for Statically Scheduled DSP Programs", *Journal of VLSI Signal Processing*, June 1996
- T.C. Denk and K.K. Parhi, "VLSI Architectures for Lattice Structure Based Orthonormal Discrete Wavelet Transforms", *IEEE Transactions on Circuits and Systems, Part - II: Analog and Digital Signal Processing*, to appear
- S. Jain and K.K. Parhi, "Efficient VLSI Architectures for Finite Field Arithmetic", *Submitted to IEEE Trans. on VLSI Systems*, April 1995
- T.C. Denk and K.K. Parhi, "Synthesis of Folded Pipelined Architectures for Multirate DSP Algorithms", *Submitted to IEEE Trans. on VLSI Systems*, November 1995
- K. Ito and K.K. Parhi, "A Generalized Technique for Register Counting and its Application to Cost-Optimal DSP Architecture Synthesis", *Submitted to Journal of VLSI Signal Processing*, Jan. 1996

- K. Ito, L.E. Lucke and K.K. Parhi, "ILP Based Cost-Optimal DSP Synthesis with Module Selection and Data Format Conversion", *Submitted to IEEE Trans. on VLSI Systems*, Feb. 1996
- Y.-N. Chang, C.Y. Wang, and K.K. Parhi, "Loop-List Allocation and Scheduling with using Heterogeneous Functional Units", *Submitted to Journal of VLSI Signal Processing*, Feb. 1996
- T.C. Denk and K.K. Parhi, "Exhaustive Scheduling and Retiming of Digital Signal Processing Systems", *Submitted to IEEE Trans. on Circuits and Systems, Part II: Analog and Digital Signal Processing*, May 1996
- T.C. Denk and K.K. Parhi, "Two-Dimensional Retiming", *Submitted to IEEE Trans. on VLSI Systems*, July 1996
- T.C. Denk, and K.K. Parhi, "Calculation of Minimum Number of Registers in 2-D Discrete Wavelet Transforms using Lapped Block Processing", *Proc. of 1994 IEEE Int. Symp. on Circuits and Systems*, pp. 3.77-3.80, May 30 - June 2, 1994, London
- K.K. Parhi and T.C. Denk, "VLSI Discrete Wavelet Transform Architectures", in *Proc. of the 1st ARPA RASSP Conference*, pp. 154-170, Aug. 15-18, 1994, Arlington (VA)
- T.C. Denk, and K.K. Parhi, "Architectures for Lattice Structure Based Orthonormal Discrete Wavelet Transforms", *Proc. of the 1994 Int. Conf. on Application Specific Array Processors*, pp. 259-270, San Francisco, August 1994
- K. Ito, L.E. Lucke and K.K. Parhi, "Module Selection and Data Format Conversion for Cost-Optimal DSP Synthesis", *Proc. of the IEEE/ACM Int. Conf. on Computer Aided Design*, pp. 322-329, Nov. 6-10, 1994, San Jose (CA)
- K. Ito and K.K. Parhi, "Determining the Iteration Bound of Data-Flow Graphs", *Proc. of the IEEE Asia-Pacific Conference on Circuits and Systems*, pp. 163-168, Dec. 5-8, 1994, Grand Hotel, Taipei

- S. Jain and K.K. Parhi, "A Low-Latency Standard Basis GF(2^m) Multiplier", in *Proc. of the 1995 IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, pp. 2747-2750, May 1995, Detroit (MI)

- C.-Y. Wang and K.K. Parhi, "MARS: A High-Level DSP Synthesis Tool Integrated within the Mentor Graphics Environment", in *Proc. of Mentor Graphics Users' Group Annual Conference*, October 22-27, 1995, Portland

- Y.N. Chang, C.Y. Wang and K.K. Parhi, "High-Level DSP Synthesis with Heterogeneous Functional Units using the MARS-II System", Proc. of the 1995 Asilomar Conf. on Signals, Systems and Computers, pp. 109-116, Pacific Grove (CA), November 1995 (*invited talk*)

- Y.-N. Chang, C.Y. Wang, and K.K. Parhi, "Loop List Scheduling for Heterogeneous Functional Units", *Proc. of Sixth Great Lakes Symp. on VLSI*, pp. 2-7, March 1996, Ames (Iowa)

- S.K. Jain and K.K. Parhi, "Efficient Standard Basis Reed-Solomon Encoder", *in Proc. of 1996 IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, pp. 3287-3290, May 1996, Atlanta

- T.C. Denk, M. Majumdar and K.K. Parhi, "Two-Dimensional Retiming with Low Memory Requirements", *in Proc. of 1996 IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, pp. 3330-3333, May 1996, Atlanta

- A. Shalash and K.K. Parhi, "Comparison of Discrete Multitone and Carrierless AM/PM Techniques for Line Equalization", *in Proc. of 1996 IEEE Int. Symp. on Circuits and Systems*, pp. II: 560-563, May 1996, Atlanta

- T.C. Denk and K.K. Parhi, "A Unified Framework for Characterizing Retiming and Scheduling Solutions", *in Proc. of 1996 IEEE Int. Symp. on Circuits and Systems*, pp. 568-571, May 1996, Atlanta

- L.L. Song and K.K. Parhi, "Efficient Finite Field Serial/Parallel Multiplication", *Proc. of the 1996 Int. Conf. on Applications-specific Systems, Architectures, and Processors*, Chicago, August 1996

- C. Xu, C.-Y. Wang and K.K. Parhi, "Order-Configurable Programmable Power-Efficient FIR Filters ", *Proc. of the 3rd International Workshop on Image and Signal Processing Advances in Computational Intelligence*, UK, November 1996

References

[1] M. C. McFarland, A. C. Parker, and R. Camposano, "The high-level synthesis of digital systems," *Proceedings of the IEEE*, vol. 78, pp. 301–318, February 1990.

[2] C.-Y. Wang and K. K. Parhi, "High-level DSP synthesis," in *Microsystems Technology for Multimedia Applications* (B. Sheu, M. Ismail, E. Sanchez-Sinencio, and T. H. Wu, eds.), ch. 8.2, pp. 615–627, IEEE Press, 1995.

[3] R. Camposano and W. Wolf, eds., *High Level VLSI Synthesis.* Kluwer Academic Publishers, 1991.

[4] M. A. Bayoumi, ed., *VLSI Design Methodologies for Digital Signal Processing Architectures.* Kluwer Academic Publishers, 1991.

[5] J. Vanhoof, I. Bolsens, G. Goosens, H. J. De Man, and K. Rompaey, *High Level Synthesis for Real-Time Digital Signal Processing.* Kluwer Academic Press, 1993.

[6] H. De Man et. al., "Architecture driven synthesis techniques for VLSI implementation of DSP algorithms," *Proceedings of the IEEE*, pp. 319–335, February 1990.

[7] L.-F. Chao, A. LaPaugh, and E. Sha, "Rotation scheduling," in *Design Automation Conference*, pp. 566–572, June 1993.

[8] T. A. Ly and J. T. Mowchenko, "Applying simulated evolution to high-level synthesis," *IEEE Transactions on Computer-Aided Design*, pp. 389–409, March 1993.

[9] C.-T. Hwang and Y.-C. Hsu, "Zone scheduling," *IEEE Transactions on Computer-Aided Design*, vol. 12, pp. 926–934, July 1993.

[10] J. Biesenack *et al.*, "The siemens high-level synthesis system callas," *IEEE Transactions on VLSI Systems*, vol. 1, September 1993.

[11] I.-C. Park and C.-M. Kyung, "FAMOS: An efficient scheduling algorithm for high-level synthesis," *IEEE Transactions on Computer-Aided Design*, vol. 12, pp. 1437–1448, October 1993.

[12] T.-F. Lee, A. C.-H. Wu, D. D. Gajski, and Y.-L. Lin, "A transformation-based method for loop folding," *IEEE Transactions on Computer-Aided Design*, vol. 13, pp. 439–450, April 1994.

[13] S. Amellal and B. Kaminska, "Functional synthesis of digital systems with TASS," *IEEE Transactions on Computer-Aided Design*, vol. 13, pp. 537–552, may 1994.

[14] C.-Y. Wang and K. K. Parhi, "High-level synthesis using concurrent transformations, scheduling, and allocation," *IEEE Transactions on Computer-Aided Design*, vol. 14, pp. 274–295, March 1995.

[15] C.-Y. Wang and K. K. Parhi, "Resource-constrained loop list scheduler for DSP algorithms," *Journal of VLSI Signal Processing*, vol. 11, pp. 75–96, October/November 1995.

[16] B. S. Haroun and M. I. Elmasry, "Architecural synthesis for DSP silicon compilers," *IEEE Transactions on Computer-Aided Design*, vol. 8, pp. 431–447, April 1989.

[17] J. Rabaey, C.-M. Chu, P. Hoang, and M. Potkonjak, "Fast prototyping of datapath-intensive architectures," *IEEE Design and Test*, vol. 8, pp. 40–51, June 1991.

[18] L. Ramachandran and D. D. Gajski, "An algorithm for component selection in performanced optimized scheduling," in *International Conference on Computer-Aided Design*, (Santa Clara, CA), pp. 92–95, November 1991.

[19] A. H. Timmer and J. A. Jess, "Execution interval analysis under resource constraints," in *International Conference on Computer-Aided Design*, pp. 454–459, November 1993.

[20] M. Ishikawa and G. De Micheli, "A module selection algorithm for high-level synthesis," in *International Symposium on Circuits and Systems*, (Singapore), pp. 1777–1780, June 1991.

[21] C.-T. Hwang, J.-H. Lee, and Y.-C. Hsu, "A formal approach to the scheduling problem in high-level synthesis," *IEEE Transactions on Computer-Aided Design*, vol. 10, pp. 464–475, April 1991.

[22] C. Hwang *et al.*, "PLS: Scheduler for pipeline synthesis," *IEEE Transactions on Computer-Aided Design*, vol. 12, pp. 1279–1286, September 1993.

[23] C. H. Gebotys and M. Elmasry, "Global optimization approach for architecture synthesis," *IEEE Transactions on Computer-Aided Design*, vol. 12, pp. 1266–1278, September 1993.

[24] C. H. Gebotys, "An optimization approach to the synthesis of multichip architectures," *IEEE Transactions on VLSI Systems*, vol. 2, pp. 11–20, March 1994.

[25] K. Ito, L. E. Lucke, and K. K. Parhi, "Module selection and data format conversion for cost-optimal DSP synthesis," in *International Conference on Computer-Aided Design*, (San Jose, CA), pp. 322–329, November 1994.

[26] K. Ito and K. K. Parhi, "Register minimization in cost-optimal synthesis of dsp architectures," in *VLSI Signal Processing VIII* (T. Nishitani and K. K. Parhi, eds.), pp. 207–216, IEEE Press, 1995. (Proc. of the 1995 IEEE Workshop on VLSI Signal Processing, Osaka, Japan).

[27] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Transactions on Computers*, vol. 36, pp. 24–35, January 1987.

[28] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman and Co., 1979.

[29] R. I. Hartley and J. R. Jasica, "Behavioral to structural translation in a bit-serial silicon compiler," *IEEE Transactions on Computer-Aided Design*, vol. 7, pp. 877–886, August 1988.

[30] K. K. Parhi and D. G. Messerschmitt, "Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding," *IEEE Trans. on Computers*, vol. 40, pp. 178–195, February 1991.

[31] M. Renfors and Y. Neuvo, "The maximum sampling rate of digital filters under hardware speed constraints," *IEEE Transactions on Circuits and Systems*, pp. 196–202, 1981.

[32] E. Reingold *et al.*, *Combinatorial Algorithms - Theory and Practice.* Prentice Hall, 1977.

[33] C. H. Gebotys and M. I. Elmasry, "Global optimization approach for architecture synthesis," *IEEE Trans. Computer-Aided Design*, vol. CAD-12, pp. 1266–1278, Sept. 1993.

[34] C. H. Gebotys and M. I. Elmasry, "Optimal synthesis of high-performance architectures," *IEEE Journal of Solid-State Circuits*, vol. 27, pp. 389–397, Mar. 1992.

[35] C.-T. Hwang, J.-H. Lee, and Y.-C. Hsu, "A formal approach to the scheduling problem in high level synthesis," *IEEE Trans. Computer-Aided Design*, vol. CAD-10, pp. 464–475, Apr. 1991.

[36] P. G. Paulin and J. P. Knight, "Force-directed scheduling for the behavioral synthesis of asic's," *IEEE Trans. Computer-Aided Design*, vol. CAD-8, pp. 661–679, June 1989.

[37] C.-Y. Wang and K. K. Parhi, "Loop list scheduler for dsp algorithms under resource constraints," in *Proc. IEEE Int. Symp. Circuits and Systems*, (Chicago), pp. 1662–1665, May 1993.

[38] C. H. Gebotys and R. J. Gebotys, "Optimal mapping of dsp applications to architectures," in *Proc. 26th Hawaii Int. Conf. System Sciences*, pp. 116–123, 1993.

[39] R. Hartley and P. Corbett, "Digit-serial processing techniques," *IEEE Trans. Circuits Syst.*, vol. CAS-37, pp. 707–719, June 1990.

[40] K. K. Parhi, "A systematic approach for design of digit-serial processing architecture," *IEEE Trans. Circuits Syst.*, vol. CAS-38, pp. 358–375, Apr. 1991.

[41] K. K. Parhi, "Systematic synthesis of dsp data format converters using life-time analysis and forward-backward register allocation," *IEEE Trans. Circuits Syst.-II: Analog and Digital Signal Processing*, vol. CAS-39, pp. 423–440, July 1992.

[42] A. Brooke, D. Kendrick, and A. Meeraus, *GAMS: A User's Guide, Release 2.25.* South San Francisco, CA: The Scientific Press, 1992.

[43] K. K. Parhi, "Algorithm transformation techniques for concurrent processors," *Proc. of the IEEE*, vol. 77, pp. 1879–1895, Dec. 1989.

[44] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow program for digital signal processing," *IEEE Trans. Computers*, vol. C-36, pp. 24–35, Jan. 1987.

[45] M. Renfors and Y. Neuvo, "The maximum sampling rate of digital filters under hardware speed constraints," *IEEE Trans. Circuits Syst.*, vol. CAS-28, pp. 196–202, Mar. 1981.

[46] D. A. Schwartz and I. T. P. Barnwell, "A graph theoretic technique for the generation of systolic implementations for shift invariant flow graphs," in *Proc. of the 1984 IEEE ICASSP*, (San Diego, CA), Mar. 1984.

[47] K. K. Parhi and D. G. Messerschmitt, "Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding," *IEEE Trans. Computers*, vol. C-40, pp. 178–195, Feb. 1991.

[48] D. Y. Chao and D. Y. Wang, "Iteration bounds of single-rate data flow graphs for concurrent processing," *IEEE Trans. Circuits Syst.-I*, vol. CAS-40, pp. 629–634, Sept. 1993.

[49] S. H. Gerez, S. M. Heemstra de Groot, and O. E. Herrmann, "A polynomial-time algorithm for the computation of the iteration-period bound in recursive data-flow graphs," *IEEE Trans. Circuits Syst.-I*, vol. CAS-39, pp. 49–52, Jan. 1992.

[50] R. M. Karp, "A characterization of the minimum cycle mean in a digraph," *Discrete Mathematics*, vol. 23, pp. 309–311, 1978.

[51] S. Y. Kung, H. J. Whitehouse, and T. Kailath, *VLSI and Modern Signal Processing.* EnglewoodCliffs, NJ: Prentice Hall, 1985.

[52] J.-G. Chung and K. K. Parhi, "Pipelining of lattice iir digital filters," *IEEE Trans. Signal Processing*, vol. SP-42, pp. 751–761, Apr. 1994.

[53] L.-F. Chao and A. LaPaugh, "Rotation scheduling: A loop pipelining algorithm," in *Proc. of ACM/IEEE Design Automation Conference*, pp. 566–572, 1993.

[54] T. C. Denk and K. K. Parhi, "A unified framework for characterizing retiming and scheduling solutions," in *Proceedings of IEEE ISCAS*, vol. 4, (Atlanta, GA), pp. 568–571, May 1996.

[55] T. C. Denk and K. K. Parhi, "Exhaustive scheduling and retiming of digital signal processing systems," *submitted to IEEE Transactions on Circuits and Systems–II: Analog and Digital Signal Processing*, May 1996.

[56] J. Monteiro, S. Devadas, and A. Ghosh, "Retiming sequential circuits for low power," in *Proceedings of IEEE Int. Conf. on Computer Aided Design*, pp. 398–402, 1993.

[57] C. Leiserson, F. Rose, and J. Saxe, "Optimizing synchronous circuitry by retiming," *Third Caltech Conference on VLSI*, pp. 87–116, 1983.

[58] S. Simon, E. Bernard, M. Sauer, and J. Nossek, "A new retiming algorithm for circuit design," in *Proceedings of IEEE ISCAS*, (London, England), May 1994.

[59] M. Potkonjak and J. Rabaey, "Retiming for scheduling," in *VLSI Signal Processing IV*, pp. 23–32, November 1990.

[60] T. C. Denk and K. K. Parhi, "Lower bounds on memory requirements for statically scheduled DSP programs," to appear in *Journal of VLSI Signal Processing*, June 1996.

[61] N. L. Passos, E. H.-M. Sha, and S. C. Bass, "Optimizing DSP flow graphs via schedule-based multidimensional retiming," *IEEE Transactions on Signal Processing*, vol. 44, pp. 150–155, January 1996.

[62] N. Passos and E. H.-M. Sha, "Full parallelism in uniform nested loops using multi-dimensional retiming," in *Proc. Int'l Conf. on Parallel Processing*, 1994.

[63] T. C. Denk and K. K. Parhi, "Two-dimensional retiming," *submitted to IEEE Transactions on VLSI Systems*, July 1996.

[64] S. G. Mallat, "Multifrequency channel decompositions of images and wavelet models," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, pp. 2091–2110, December 1989.

[65] I. Daubechies, "Orthonormal bases of compactly supported wavelets," *Comm. in Pure and Applied Math.*, vol. 41, pp. 909–996, November 1988.

[66] G. Strang, "Wavelets and dilation equations: A brief introduction," *SIAM Rev.*, vol. 31, pp. 614–627, December 1989.

[67] M. Vetterli and C. Herley, "Wavelets and filter banks: Theory and design," *IEEE Transactions on Signal Processing*, vol. 40, pp. 2207–2232, September 1992.

[68] O. Rioul and M. Vetterli, "Wavelets and signal processing," *IEEE Signal Processing Magazine*, pp. 14–38, October 1991.

[69] C. Chakrabarti, M. Vishwanath, and R. Owens, "Architectures for wavelet transforms," in *Proceedings of IEEE ICASSP*, (Detroit, MI), 1995.

[70] K. K. Parhi, C.-Y. Wang, and A. P. Brown, "Synthesis of control circuits in folded pipelined DSP architectures," *IEEE Journal of Solid-State Circuits*, vol. 27, pp. 29–43, January 1992.

[71] K. K. Parhi, "A systematic approach for design of digit-serial signal processing architectures," *IEEE Transactions on Circuits and Systems*, vol. 38, pp. 358–375, April 1991.

[72] G. Knowles, "VLSI architecture for the discrete wavelet transform," *Electronics Letters*, vol. 26, pp. 1184–1185, July 1990.

[73] K. K. Parhi and T. Nishitani, "VLSI architectures for discrete wavelet transforms," *IEEE Transactions on VLSI Systems*, vol. 1, pp. 191–202, June 1993.

[74] C. Chakrabarti and M. Vishwanath, "Efficient realizations of the discrete and continuous wavelet transforms: From single chip implementations to mappings on SIMD array computers," *IEEE Transactions on Signal Processing*, vol. 43, pp. 759–771, March 1995.

[75] T. C. Denk and K. K. Parhi, "Systematic design of architectures for M-ary tree-structured filter banks," in *VLSI Signal Processing, VIII* (T. Nishitani and K. Parhi, eds.), pp. 157–166, IEEE Press, October 1995.

[76] T. C. Denk and K. K. Parhi, "Synthesis of folded pipelined architectures for multirate DSP algorithms," *submitted to IEEE Transactions on VLSI Systems*, November 1995.

[77] T. C. Denk and K. K. Parhi, "Architectures for lattice structure based orthonormal discrete wavelet transforms," in *Proc. of 1994 IEEE International Conf. on Application-Specific Array Proc.*, (San Francisco, CA), pp. 259–270, IEEE Computer Society Press, August 1994.

[78] K. K. Parhi and T. C. Denk, "VLSI discrete wavelet transform architectures," in *Proceedings of First Annual RASSP Conference*, (Arlington, VA), pp. 154–170, August 1994.

[79] T. C. Denk and K. K. Parhi, "VLSI architectures for lattice structure based orthonormal discrete wavelet transforms," to appear in *IEEE Transactions on Circuits and Systems–II: Analog and Digital Signal Processing.*

[80] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-power CMOS digital design," *IEEE Journal of Solid-State Circuits*, vol. 27, pp. 473–484, April 1992.

[81] K. K. Parhi, "Systematic synthesis of DSP data format converters using life-time analysis and forward-backward register allocation," *IEEE Transactions on Circuits and Systems–II: Analog and Digital Signal Processing*, vol. 39, pp. 423–440, July 1992.

[82] L. Stok and J. Jess, "Foreground memory management in data path synthesis," *Interational Journal of Circuit Theory and Applications*, vol. 20, pp. 235–255, 1992.

[83] J. Bae, V. Prasanna, and H. Park, "Synthesis of a class of data format converters with specified delays," in *Proceedings of 1994 IEEE International Conference on Application-Specific Array Processors*, (San Francisco, CA), pp. 283–294, IEEE Computer Society Press, August 1994.

[84] C.-Y. Wang and K. K. Parhi, "High-level DSP synthesis using concurrent transformations, scheduling, and allocation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, pp. 274–295, March 1995.

[85] P. P. Vaidyanathan, *Multirate Systems and Filter Banks.* Englewood Cliffs, NJ: Prentice Hall, 1993.

[86] A. K. Soman and P. P. Vaidyanathan, "On orthonormal wavelets and paraunitary filter banks," *IEEE Transactions on Signal Processing*, vol. 41, pp. 1170–1183, March 1993.

[87] P. P. Vaidyanathan and P. Hoang, "Lattice structures for optimal design and robust implementation of two-channel perfect reconstruction QMF banks," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-36, pp. 81–94, January 1988.

[88] R. Hartley and P. Corbett, "Digit-serial processing techniques," *IEEE Transactions on Circuits and Systems*, vol. 37, pp. 707–719, June 1990.

[89] S. G. Smith and P. B. Denyer, *Serial Data Computation.* Boston, MA: Kluwer Academic, 1988.

[90] R. Coifman and M. Wickerhauser, "Entropy-based algorithms for best basis selection," *IEEE Transactions on Information Theory*, vol. 38, pp. 713–718, March 1992.

[91] J. W. Lechleider, "High Bit Rate Digital Subscriber Lines: A Review of HDSL Progress," *IEEE J-SAC.*, vol. 9, pp. 769–784, Aug. 1991.

[92] P.S. Chow *et al.*, "Performance Evaluation of a Multichannel Transceiver System for ADSL and VHDSL Services," *IEEE J-SAC.*, vol. 9, pp. 909–919, Aug. 1991.

[93] D. W. Lin, C.-T. Chen, and T. R. Hsing, "Video On Phone Lines," *Proc. IEEE*, vol. "83(2)", pp. 175–193, 1995.

[94] G. H. Im and J-J. Werner, "Bandwidth Efficient Digital Transmission up to 155 Mb/s Over Unshielded Twisted-Pair Cables," *IEEE Conf. on Commun.*, vol. 3, pp. 1797–1803, 1993.

[95] J. Chow, J. Tu, and J. Cioffi, "A Discrete Multitone Transceiver System for HDSL Applications," *IEEE J-SAC*, vol. 9, pp. 909–919, 1991.

[96] I. Kalet, "The multitone channel," *IEEE Transactions on Communication*, vol. 37, no. 2, pp. 119–124, 1989.

[97] J. Bingham, "Multicarrier Modulation for Data Transmission: An Idea whose time Has Come," *IEEE Comm. Magazine*, vol. 28, pp. 5–14, May 1990.

[98] G-H. Im *et al.*, "51.84 Mb/s 16-CAP ATM LAN standard," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 4, pp. 620–623, 1995.

[99] B.R. Petersen and D.D. Falconer, "Minimum mean square equalization in cyclostationary and stationary interference- analysis and subscriber line calculations," *IEEE J-SAC*, vol. 9, pp. 931–940, Aug. 1991.

[100] N. Shanbhag and K. K. Parhi, "Pipelined adaptive digital filters," *Kluwer*, 1994.

[101] D. Harman *et al.*, "Local Distribution for IMTV," *IEEE Multimedia*, vol. 2, number 3, Fall 1995.

[102] P. P. Vaidyanathan, *Multirate Systems and Filter Bank.* Prentice Hall, 1993.

[103] R. K. Brayton *et al.*, "A New Algorithm for Statistical Circuit Design Based on Quasi-Newton Methods and Function Splitting," *IEEE Transactions on Circuits and Systems.*, vol. 26, pp. 784–794, 1979.

[104] S. K. Jain and K. K. Parhi, "Efficient power based Galois Field arithmetic architectures," in *IEEE Workshop on VLSI Signal Processing*, (San Diego), pp. 306–316, Oct. 1994.

[105] S. K. Jain and K. K. Parhi, "Low Latency standard basis GF(2^m) multiplier and squarer architectures," in *Proc. IEEE ICASSP*, (Detroit (MI)), pp. 2747–2750, May 1995.

[106] S. K. Jain and K. K. Parhi, "Efficient Standard Basis Reed-Solomon Encoder," in *Proc. of 1996 IEEE Int. Conf. of Acoustics, Speech, and Signal Processing*, (Atlanta), May 1996.

[107] L. Song and K. K. Parhi, "Efficient Finite Field Serial/Parallel Multiplication," in *Proc. of International Conf. on Application Specified Systems, Architectures and Processors*, (Chicago), Aug 1996.

[108] N. Weste and K. Eshranghiam, *Principle of CMOS VLSI Design.* Addison-Wesley Publishing Company, 1992.

[109] C. C. W. et al, "VLSI Architectures for Computing Multiplications and Inverses in GF(2^m)," *IEEE Trans. on Computers*, vol. c-34, pp. 709–716, August 1985.

[110] C. L. Wang, "Bit-Level Systolic Array for Fast Exponentiation in GF(2^m)," *IEEE Trans. on Computers*, vol. 43, pp. 838–841, July 1994.

[111] G. Feng, "A VLSI architecture for fast inversion in $GF(2^m)$," *IEEE Trans. on Computers*, vol. 38, pp. 1383–1386, Oct 1989.

[112] I. S. Hsu, T. K. Truong, L. J. Deutsch, and I. S. Reed, “A Comparison of VLSI Architecture of Finite Field Multipliers using Dual, Normal, or Standard Bases,” *IEEE Trans on Computers*, vol. 37, pp. 735–739, June 1988.

[113] J. Yuan and C. Svensson, “High-speed CMOS circuit techniques,” *IEEE Journal of Solid-State Circuits*, vol. 24, pp. 62–70, Feb 1989.

[114] A. Salz and M. Horowitz, “IRSIM: An incremental MOS switch-level simulator,” in *Proc. of 26th ACE/IEEE Design Automation Conf.*, (1989), pp. 173–178, June.

[115] C.-S. Yeh, I. S. Reed, and T. K. Truong, “Systolic Multipliers for Finite Fields GF(2^m),” *IEEE Trans. on Computers*, vol. c-33, pp. 357–360, April 1984.

[116] E. R. Berlekamp, “Bit serial Reed-Solomon encoders,” *IEEE Trans on information Theory*, vol. IT-28, pp. 869–874, Nov. 1982.

[117] S. W. Wei, “A Systolic Power-Sum Circuit for GF(2^m),” *IEEE Trans. on Computers*, vol. 43, pp. 226–229, Feb 1994.

[118] R. E. Blahut, *Theory and Pratice of Error Control Codes.* Addison Wesley, 1984.

[119] C. L. Wang and J. L. Lin, “Systolic Array Implementation of Multipliers for Finite Field GF(2^m),” *IEEE Trans. on Circuits and Systems*, vol. 38, pp. 796–800, July 1991.

[120] M. A. Hasan and V. K. Bhargava, “Division and bit-serial multiplication over GF(q^m),” *IEE Proceedings-E*, vol. 139, pp. 230–236, May 1992.

[121] P. A. Scott, S. E. Tavares, and L. E. Peppard, “A Fast VLSI Multiplier for GF(2^m),” *IEEE Journal on Selected ares in Communications*, vol. SAC-4, pp. 62–66, Jan. 1986.

[122] J. H. Satyanarayana and K. K. Parhi, “HEAT: Hierarchical Energy Analysis Tool,” in *Proc. 33rd ACM/IEEE Design Automation Conf.*, (Las Vegas), pp. 9–14, June 1996.

[123] A. P. Chandrakasan and R. W. Brodersen, “Minimizing power consumption in digital CMOS circuits,” *Proceedings of the IEEE*, vol. 83, pp. 498–523, April 1995.

[124] D. Singh, J. M. Rabaey, M. Pedram, F. Catthoor, S. Rajgopal, N. Sehgal, and T. J. Mozdzen, “Power conscious CAD tools and methodologies: A perspective,” *Proceedings of the IEEE*, vol. 83, pp. 570–, April 1995.

[125] A. P. Chandrakasan and R. W. Brodersen, “Design of portable systems,” in *IEEE Custom Integrated Circuits Conference*, (San Diego, CA), pp. 259–266, May 1994.

[126] S. D. Brown, “An overview of technology, architecture and CAD tools for programmable logic devices,” in *IEEE Custom Integrated Circuits Conference*, (San Diego, CA), pp. 69–76, May 1994.

[127] V. Visvanathan and S. Ramanathan, “Synthesis of Energy-Efficient Configurable Processor Arrays,” in *International workshop on Parallel Processing*, 1994.

[128] K. Parhi, C.Wang, and A.P.Brown, “Synthesis of control circuits in folded pipelined architecture s,” *IEEE J. Solid State Circuits*, vol. 27, pp. 29–43, Jan 1992.

[129] C.E.Leiserson and J. Saxe, "Optimizing synchronous systems," in *VLSI and Computer Systerms*, pp. 41–67, 1983.

[130] I. Koren, *Computer Arithmetic Algorithms.* Prentice-Hall, 1993.

[131] C. S. Wallace, "A suggestion for a fast multiplier," *Computer Arithmetic*, vol. 1, pp. 114–117, 1990.

[132] N. H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design: A Systems Prospective.* Addison-Wesley Publishing Company, 2nd ed., 1993.